

Automatic Test Generation for Data-Flow Reactive Systems Modeled by Variable Driven Timed Automata

Omer Nguena Timo¹, Hervé Marchand², and Antoine Rollet¹

¹ LaBRI (University of Bordeaux - CNRS)
Talence, France,

{nguena, rollet}@labri.fr,

² INRIA, Centre Rennes-
Bretagne Atlantique
herve.marchand@inria.fr,

Abstract. In this paper, we handle the problem of conformance testing for data-flow critical systems with time constraints. We present a formal model (Variable Driven Timed Automata) adapted for such systems inspired from timed automata using variables as inputs and outputs, and clocks. In this model we consider urgency and the possibility to fire several transitions instantaneously. We present a conformance relation for this model and we propose a test generation method using a test purpose approach. This method is illustrated with an example on a "Bi-manual command".

1 Introduction

Testing is one of the most popular techniques used to increase the quality of a software. Since systems are getting more and more complex, formal approaches permit to obtain efficient and rigorous testing frameworks. Testing is a large domain since many characteristics may be focused, such as conformance, performance, interoperability or robustness... In this paper we consider formal conformance testing, i.e. checking if the observable behaviour of an implementation conforms to its specification described in a formal model.

Testing techniques may be very different depending on the kind of systems we intend to validate (e.g. embedded systems, communication protocols, etc...). In this work, we handle testing for critical data-flow reactive systems with time constraints. Such systems are widely used in the industrial automation domain. Data-flow reactive systems are characterized by the fact that they interact with their environment in a continuous way by means of continuous input and output set of events (taking their values in (possibly) infinite domains), while obeying some timing constraints. In this framework, continuous means that the values of the inputs events can be updated at anytime, while the value of the outputs events can always be observed. Their particularity makes that models used in testing methods (e.g. Labelled Transition Systems with inputs, outputs and/or

time) are not well adapted to describe such systems as they do not include data-flow synchronous aspects with dense time.

Related work. Timed automata [1] have been a reference model for many testing approaches for Real-time systems. [4] and [14] propose methods based on characterization of states inspired from Finite State Machines (FSM) theory applied on extensions of timed automata with inputs and outputs. [6] and [14] use the region graph ([1]) as a basis for test case generation.

The LTS/ioco theory ([15]) has also inspired many testing approaches for timed systems. [11] defines new conformance relations on Timed Extended Finite State Machines (TEFSM), a timed extension of FSM. Using the LTS semantics, [7] proposes a timed extension of the *ioco* relation (*tioco*) which includes delays in the set of observable outputs. They propose two kinds of tests : one with analog-clock (dense time) and one with digital-clock using a special “*tick*” action in the model. The Uppaal project ([9]) uses timed automata with variables and urgency to model the systems and uses a symbolic extension of *ioco* to generate test cases. As the previously mentioned methods, it uses an event based semantics not adapted for data-flow synchronous systems.

Many testing methods have also been proposed for data-flow synchronous systems ([3], [12], [8]). Lutess ([3], [13]) is a testing environment based on Lustre ([5]) using a “Lustre-like” model of the environment to lead test data generation. The recent version of Lutess uses Constraint Logic Programming. The Lurette environment ([12]) is similar to Lutess but uses ad-hoc notations to describe the environment. The Gatel tool ([8]) generates test cases with a white box approach : it translates the program and its environment specification into a system of constraints description and uses it to generate test data according to a test objective. All these synchronous approaches are widely used in the industry. However they do not permit to handle dense time. Timed Automata with urgent transitions, as studied in [2], allow shorter and clear specifications and it has been proved that from a language theoretic point of view, addition of urgency does not improve the expressive power of timed automata. But model in [2] is (discrete) event-based and it is not adequate for data-flow systems. To our knowledge, no approach has been proposed in the literature combining data-flow synchronous aspects and dense time.

Contributions. In this paper, we introduce the Variable Driven Timed Automata (VDTA) model, a variant of timed automata [1] with variables, which permits to describe data-flow systems with dense time. As in [9], transitions are urgent but our model permits to fire several transitions instantaneously as long as the guard is satisfied. The inputs and the outputs of the system are variables : the tester can assign new values to the input variables and observes the output ones. In the semantics of VDTA, we consider two kinds of transitions :

- *delay* transitions for time elapsing
- *discrete* transitions including *urgent* transitions when the guard is satisfied, and *input-update* transitions when the value of an input variable changes but the guard is still not satisfied.

From a testing point of view, we assume that the specification of the system under test is modeled by a VDTA. Our aim is then to derive a tester allowing to

check whether an implementation (that could also be modeled by an unknown VDTA) conforms with its specification. Roughly, an implementation conforms to its specification whenever after an observable sequence of input updates and delays, the values of the output variables of the implementation are the same as the ones of the specification after the same sequence.

In order to limit the number of test cases, we propose a generation method based on a selection by a test purpose allowing to target some particular behaviors of the specification that we want to test on the implementation. We define test purposes as VDTAs equipped with a set of accepting locations playing the role of a non intrusive observer. It amounts to perform a product between the specification and the test purpose and to select the behavior leading to some particular configurations of this synchronous product (the ones that reach the accepting states of the TP). The last point is achieved by performing a coreachability analysis on a variant of the region graph derived from the VDTA and adding new constraints on the guards.

Organization of the paper. The structure of the document is as follows : Section 2 presents some definitions and notations concerning the VDTA model and its semantics. In Section 3, we introduce the notion of *Time Abstract Graph* and we outline the problem of reachability analysis on VDTA. Section 4 presents the conformance relation called *tvco*, as well as the synchronous product between two VDTAs that will be used to combine the specification and the test purpose and finally describes the symbolic test case generation methodology.

2 Model and notations

In this section, we present Variable Driven Timed Automata (VDTA), a variant of timed automata extended with data, urgency and synchronous data-flow aspects : the model gives the possibility to fire several transitions (in case of successive true guards) in null delay. We also give the corresponding semantics.

2.1 Variable driven timed automata definition

Variables, Assignments, Constraints. Let \mathbb{N} , \mathbb{Q}_+ and \mathbb{R}_+ denote the sets of natural, non-negative rationals and real numbers, respectively. Let $V = \{V_1, \dots, V_n\}$ be a set of variables; each variable $V_i \in V$ ranges over a (possibly infinite) domain $Dom(V_i)$ in \mathbb{N} , \mathbb{Q}_+ or \mathbb{R}_+ . We define $Dom(V) = \prod_{i \in [1..n]} Dom(V_i)$, the domain of V . In the sequel, v_i denotes a valuation of the variable V_i and v the tuple of valuations of the set of variables V . A variable assignment for V is a tuple $\prod_{i \in [1..n]} (\{V_i\} \times (Dom(V_i) \cup \{\perp\}))$ and we denote by $A(V)$ the set of variable assignments for V . Given a valuation $v = (v_1, \dots, v_n)$ of V and a variable assignment $A \in A(V)$, we define the tuple of valuations $v[A]$ as $v[A](V_i) = c$ if (V_i, c) is an element of A and $c \neq \perp$, and $v[A](V_i) = v_i$ otherwise. Intuitively, an element (V_i, c) of variable assignment A , requires to assign c to the variable V_i if c is a constant from $Dom(V_i)$; otherwise c is equal to \perp and *no access* to the variable V_i should be done. $Var(A)$ denotes the set of variables of V that are updated by A . We denote Id_V the identity variable assignment that let unchanged all the variables of V . We denote by $\mathcal{G}(V)$ the set of variable constraints defined as boolean

combinations of simple constraints of the form $V_i \bowtie c$ with $V_i \in V$, $c \in \text{Dom}(V_i)$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Given $G \in \mathcal{G}(V)$ and a valuation $v \in \text{Dom}(V)$, we write $v \models G$ when $G(v) \equiv \text{true}$. We denote $\text{Proj}_{V_i}(G) \in \mathcal{G}(V \setminus \{V_i\})$ the constraint such that $(v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n) \models \text{Proj}_{V_i}(G)$ if and only if there exists $v_i \in \text{Dom}(V_i)$ such that $(v_1, \dots, v_n) \models G$. We extend in a natural way this projection to a subset V' of V and we denote it $\text{Proj}_{V'}(G)$.

Let $X = \{X_1, \dots, X_k\}$ be a set of *clocks*. A *clock valuation* for X is a function from X to \mathbb{R}_+ . Given X , the set of valuations is denoted \mathbb{R}_+^X . Given a valuation $x = (x_1, \dots, x_k) \in \mathbb{R}_+^X$ and $t \in \mathbb{R}_+$, $x + t$ stands for $(x + t)(X_i) = x_i + t$ for any $X_i \in X$. Let $X' \subseteq X$, $x[X' \leftarrow 0]$ is the valuation defined by $x[X' \leftarrow 0](X_i) = 0$ for any $X_i \in X'$ and $x[X' \leftarrow 0](X_i) = x_i$ otherwise. We denote by $\mathcal{G}(X)$ the set of clock constraints defined as boolean combinations of simple constraints of the form $X_i \bowtie c$ with $X_i \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Given $G_X \in \mathcal{G}(X)$ and $x \in \mathbb{R}_+^X$, we write $x \models G_X$ when $G(x) \equiv \text{true}$.

VDTA and Semantics. Variable Driven Timed Automata (VDTA) is a variant of timed automata. The main difference with timed automata with variable is that actions are assignments of variables and constraints are defined over clocks and variables. The particularity of this model is that all transitions are urgent, meaning that they must be fired as soon as guards are satisfied.

Definition 1 (VDTA). A Variable Driven Timed Automaton (VDTA) is a tuple $\mathcal{A} = \langle L, X, I, O, l^0, G^0, \Delta_{\mathcal{A}} \rangle$, where

- L is a finite set of locations, $l^0 \in L$ is the initial location,
- $X = \{X_1, X_2, \dots, X_k\}$ is a finite set of clocks,
- $I = \{I_1, I_2, \dots, I_n\}$ is a finite set of input variables,
- $O = \{O_1, O_2, \dots, O_m\}$ is a finite set of output variables,
- $G^0 \in \mathcal{G}(I, O)$ is the initial condition, a constraint with variables in $I \cup O$.
- $\Delta_{\mathcal{A}} \subseteq L \times \mathcal{G}(I, O, X) \times A(O) \times 2^X \times L$ is the transition relation:
 - $\langle l, G, A, \mathcal{X}, l' \rangle \in \Delta_{\mathcal{A}}$ is a transition such that
 - l and l' are the source and the target locations of the transition.
 - G is a boolean combination of elements of $\mathcal{G}(I)$, $\mathcal{G}(O)$ and $\mathcal{G}(X)$.
 - An output assignment $A \in A(O)$.
 - $\mathcal{X} \in 2^X$ is a set of clocks that are reset when triggering the transition.

In the sequel, we write $l \xrightarrow{G, A, \mathcal{X}} l'$ when $\langle l, G, A, \mathcal{X}, l' \rangle \in \Delta_{\mathcal{A}}$ and $G_{\mathcal{A}}(l) = \{G \in \mathcal{G}(I, O, X) \mid \exists l' \in L, l \xrightarrow{G, A, \mathcal{X}} l'\}$. The environment of a system modeled by a VDTA observes all the variables. The set I of input variables represents the variables to which the environment (e.g. the tester) can assign a value whereas the set O of output variables represents the variables for which the values are updated by the system while triggering a transition. Furthermore, we assume that all the transitions are urgent, meaning that as soon as the guard of a transition is satisfied, the transition is triggered. We also assume that the assignment of new values to the input variables is performed instantaneously. Finally, note that in each location the environment can choose any value for the input variables.

Remark 1. One can also add invariants w.r.t. the input variables in locations in order to model some environment constraints.

Example 1. We illustrate the previous definition with the following example describing the behavior of a Bi-manual command system [10]. 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 1 time unit. If only one button is pushed (then L or R is true) and a delay of 1 time unit is performed (time-out has occurred), then the whole process must be started again. 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).

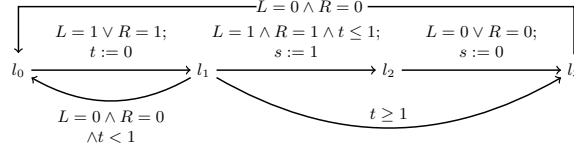


Fig. 1. The "Bi-manual command" Example

Figure 1 represents the VDTA model of this system. The model has two boolean input variables (L , R), one boolean output variable (s) and a clock (t). 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 instantaneously 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 $R = 1$, then in order to reach l_2 or l_0 , it must leave l_1 strictly before 1 time unit, otherwise, the system moves instantaneously to the location l_3 . This is possible since transitions are urgent.

These two use cases illustrate the utility of this new formalism : such behaviours are not natural to describe in usual event based models, even with Uppaal ([9]). Our attempts to model this system with UPPAAL 4.0 failed as invariants are bounded constraints and timing constraints are not allowed on urgent transitions which does not ease to model the urgency that can happen in l_1 .

Definition 2 (deterministic VDTA). A VDTA $\mathcal{A} = \langle L, X, I, O, l^0, G^0, \Delta_{\mathcal{A}} \rangle$ is deterministic if

- the initial condition G^0 is satisfied by at most one valuation (i^0, o^0), and
- for all $l \in L$, for all $G, G' \in G_{\mathcal{A}}(l)$ s.t. $G \neq G'$, $G \cap G'$ is unsatisfiable.

In the reminder of this paper, we shall only consider deterministic VDTAs.

The semantics of a VDTA is presented in terms of timed transition systems (TTS).

Definition 3. The semantics of a VDTA $\mathcal{A} = \langle L, X, I, O, l^0, G^0, \Delta_{\mathcal{A}} \rangle$, is a TTS defined by the tuple $\llbracket \mathcal{A} \rrbracket = \langle S, s^0, \Sigma, \rightarrow \rangle$ where

- $S = L \times Dom(I) \times Dom(O) \times \mathbb{R}_+^X$ is the (infinite) set of states,
- $s^0 = (l^0, i^0, o^0, x^0)$ is the initial configuration where x^0 is the clock valuation that maps every clock to 0 and (i^0, o^0) is the only solution of G^0 ,
- $\Sigma = A(I) \cup A(O) \cup \mathbb{R}_+^X$ is the (infinite) set of actions, and
- \rightarrow is the transition relation with the following three types of transitions:
 - T1** $(l, i, o, x) \xrightarrow{A} (l', i, o[A], x[\mathcal{X} \leftarrow 0])$ if there exists $(l, G, A, \mathcal{X}, l') \in \Delta_{\mathcal{A}}$ such that $(i, o, x) \models G$,
 - T2** $(l, i, o, x) \xrightarrow{A} (l, i[A], o, x)$ with $A \in A(I)$ if $\forall (l, G, A', \mathcal{X}, l') \in \Delta_{\mathcal{A}}, (i, o, x) \not\models G$.
 - T3** $(l, i, o, x) \xrightarrow{\delta} (l, i, o, x + \delta)$ with $\delta > 0$ if for every $\delta' < \delta$, for every symbolic transition $(l, G, \mathcal{X}', l') \in \Delta_{\mathcal{A}}$, we have $(i, o, x + \delta') \not\models G$.

The semantics considers two kinds of transitions: *discrete transitions* (T1 and T2) and *delay transitions* (T3). They concern the update of either input or output variables. There are two sorts of discrete transitions: *urgent transitions* (T1) and *input-update transitions* (T2). Delay transitions (T3) represent the elapse of time. Urgent transitions (T1) are fired as soon as constraints are satisfied by the current configuration of the system. Input-update transitions (T2) only allow to change the values of input variables; they are fired when the environment chooses to update them and when the guards are not satisfied. Input-update transitions and delay transitions are fired only when no urgent transition can be fired. Compared with the model in [2], in our model, events (input-update) from the environment are not explicitly specified. This make VDTA specification shorter and clearer.

Notations. When necessary, we denote \rightarrow_{T_i} for transitions of type $T_i, i = 1..3$. Given a state $s = (l, i, o, x) \in S$, $Out(s) = o$ gives access to the output value of $\llbracket \mathcal{A} \rrbracket$ in state s . We write $s \xrightarrow{a}$ when there exists s' such that $s \xrightarrow{a} s'$. For a sequence $\sigma = a_1.a_2.\dots.a_{k-1}.a_k$ of Σ^* , $s \xrightarrow{\sigma} s'$ if there exists $\{s_i\}_{i=1..k-1}$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_{k-1}} s_{k-1} \xrightarrow{a_k} s'$ and we write $s \xrightarrow{\sigma}$ if there exists s' such that $s \xrightarrow{\sigma} s'$. Given a state s of $\llbracket \mathcal{A} \rrbracket$, a run is a sequence of alternating states and actions $s = s_0.a_1.s_1 \dots a_n.s_n$ in $S.(\Sigma.S)^*$ such that $\forall i \geq 0, s_i \xrightarrow{a_{i+1}} s_{i+1}$. $Run(s, \llbracket \mathcal{A} \rrbracket)$ denotes the set of runs that can be executed in $\llbracket \mathcal{A} \rrbracket$ starting in state s and we let $Run(\llbracket \mathcal{A} \rrbracket) = Run(s^0, \llbracket \mathcal{A} \rrbracket)$. The *trace* $\rho(r)$ of a run $r = s_0.a_1.s_1 \dots a_n.s_n$ is given by the sequence $\rho(r) = Proj_S(r) = a_1 \dots a_n \in \Sigma^*$. $Tr(\llbracket \mathcal{A} \rrbracket) = \{\rho(r) | r \in Run(\llbracket \mathcal{A} \rrbracket)\}$ is the set of traces generated by \mathcal{A} .

Example 2. Back to Example 1, some possible runs derived from $\llbracket \mathcal{A} \rrbracket$ are

- $(l_0, (0, 0, 0, 0)) \xrightarrow{L:=1} (l_0, (1, 0, 0, 0)) \xrightarrow{Id_O} (l_1, (1, 0, 0, 0)) \xrightarrow{0.3} (l_1, (1, 0, 0, 0.3))$
- $(l_0, (0, 0, 0, 0)) \xrightarrow{L:=1, R:=1} (l_0, (1, 1, 0, 0)) \xrightarrow{Id_O} (l_1, (1, 1, 0, 0)) \xrightarrow{s:=1} (l_2, (1, 1, 1, 0))$

We now define the classic *Pred* operator of a set of states Q : $Pred(Q) = \{s \in S \mid \exists s' \in Q, a \in \Sigma, s \xrightarrow{a} s'\}$. Note that $Pred : 2^S \rightarrow 2^S$ is monotonic. We also define $Pre^0(Q) = Q$ and for $i \geq 0$, $Pred^{i+1}(Q) = Pred(Pred^i(Q))$. We consider the *CoReach*(\cdot) operation allowing to compute the states from which a state in Q can be reached: $CoReach(Q) = \mu X. Q \cup pre(X)$. Following [?], we have that $CoReach(Q) = \bigcup_{i \geq 0} Pred^i(Q)$.

Stable VDTA and Observed runs. In a VDTA, all transitions are urgent and several transitions can be triggered in null delay. We then consider *stable states* that are states from which no urgent transition can be fired. Formally a state s of $\llbracket \mathcal{A} \rrbracket$ is *stable* whenever for every $A \in A(O)$, $s \not\stackrel{A}{\rightarrow}$. A *stable run* is a run that ends in a stable state. To leave this state, either the input values need to be updated or we need to let the time elapse. A VDTA \mathcal{A} is *stable* if there is no loop of unstable states in $\llbracket \mathcal{A} \rrbracket$.

In the context of VDTA, a test activity consists in executing on the implementation a sequence in $(A(I) \cup A(O) \cup \mathbb{R}_+)^*$, and in checking whether the output values of the implementation coincide with those in the last state of the specification after the sequence being executed. It is worth noticing that on the implementation (seen as a VDTA) many variations on outputs can occur in zero time unit and these output changes can not be observed. Thus in our testing framework, we will assume that outputs are observed only when the implementation is in *stable* states. Given a stable state s , we will thus be interested in the next stable state (recall that VDTA are deterministic) the implementation can reach from s after the execution of an input-update $A_i \in A(I)$ followed by a sequence in $(A(O) \cup \mathbb{R}_+)^*$. This leads us to introduce the notion of observed runs. Given two stable states $s, s' \in S$, we write:

- $s \xrightarrow{A_i} s'$ if there exists a sequence $\sigma = \sigma_1 \cdots \sigma_n \in (A(O) \cup \{0\})^*$ such that $s \xrightarrow{A_i} s'' \xrightarrow{\sigma} s'$, i.e. s' is the unique stable state that can be reached from s after updating the input variables with A_i , only triggering urgent transitions in zero time unit.
- $s \xrightarrow{\delta} s'$ if there exists a sequence $\sigma = \sigma_1 \cdots \sigma_n \in (\{Id_O\} \cup \mathbb{R}_+)^*$ such that $s \xrightarrow{\sigma} s'$ and $\delta = \sum_{\delta_i \in Proj_O(\sigma)} \delta_i$, i.e. s' is the stable state that can be reached by letting the time elapse during δ units of time with no output update.

Let us denote by $Obs(\mathcal{A}) = (S, s^0, A(I) \cup \mathbb{R}_+, \Longrightarrow)$ the TTS inductively generated from $\llbracket \mathcal{A} \rrbracket$ starting from s^0 (that is supposed to be stable) using the two previous rules. The set of observed runs of \mathcal{A} is then given by the set $ObsRun(\mathcal{A}) = Run(Obs(\mathcal{A}))$, whereas the set of observed traces is given by $ObsTr(\mathcal{A}) = Tr(Obs(\mathcal{A}))$. Finally, we define s *Safter* $\alpha = \{s' \mid s \xrightarrow{\alpha} s'\}$ ³ and \mathcal{A} *Safter* $\alpha = s^0$ *Safter* α .

3 Time-Abstract Graph and reachability analysis

The *reachability analysis* amounts to checking whether, from the initial state, we can reach a target state (or location). We provide a symbolic backward reachability analysis for VDTA. The algorithm iteratively computes (urgent, input-update, time-elapsing) predecessors of states using a new representation of VDTA called *time abstract graph*. A Time abstract graph decomposes the clock constraints into *atomic* clock constraints simplifying the computation of time-elapsing predecessors as transitions are urgent.

³ *Safter* stands for “after stabilization”.

3.1 Time-Abstract graph construction

From a VDTA, one can build a *time abstract graph* whose transitions are of two sorts: urgent transitions (**U1**) and time-elapsing urgent transitions (**U2**). Time-elapsing urgent transitions correspond to atomic timing context changing and urgent transitions allow to change the contents of output variables. Time-elapsing transitions are labelled with the special action Id_O that let unchange the value of the output variables. The construction of time-abstract graphs is based on the standard notion of *region* [1] introduced for the reachability analysis of timed automata. We assume the reader is familiar with the *region construction* of [1] for timed automata. For the sake of completeness, we recall here the main definitions and properties we will use further.

Let $X = \{X_1, X_2, \dots\}$ be a finite set of clocks. Recall that the value of each clock $X_i \in X$ is denoted by x_i . For $x_i \in \mathbb{R}_+$, $\lfloor x_i \rfloor$ and $\langle x_i \rangle$ denote the integer part and the fractional part of x_i , respectively.

Definition 4 (Clock Region). *We consider a constant $K \in \mathbb{N}$. A clock region is an equivalence class of the relation \simeq_K over clock valuations. For two valuations $x, x' \in \mathbb{R}_+^X$, we have $x \simeq_K x'$ iff the following conditions hold:*

1. $\forall X_i \in X, x_i \leq K \Leftrightarrow x'_i \leq K$,
2. $\forall X_i \in X, x_i \leq K \Rightarrow (\lfloor x_i \rfloor = \lfloor x'_i \rfloor \text{ and } \langle x_i \rangle = 0 \Leftrightarrow \langle x'_i \rangle = 0)$,
3. $\forall X_i, X_j \in X, x_i \leq K \text{ and } x_j \leq K \Rightarrow (\langle x_i \rangle \leq \langle x_j \rangle) \Leftrightarrow \langle x'_i \rangle \leq \langle x'_j \rangle$.

We let $Reg_K(X)$ be the set of clock regions for a constant K . We recall that the size of $Reg_K(X)$ is in $2^{O(m \cdot \log Km)}$ where $m = |X|$ (see [1]). When the constant K is clear from the context, we denote by $[x]$ the clock region that contains x , and by $\llbracket r \rrbracket$ the set of clock valuations whose clock region is equal to r . We say that a region r' is a *successor* of r and we write $r' \in Succ(r)$ if there are $v \in \llbracket r \rrbracket$, $v' \in \llbracket r' \rrbracket$ and $\delta \in \mathbb{R}_+$ such that $v' = v + \delta$. A region r' is the *immediate successor* of r , and we write $r' = I_Succ(r)$, if $r' \in Succ(r) \setminus \{r\}$ and there is no region $r'' \in Succ(r) \setminus \{r, r'\}$ such that $r' \in Succ(r'')$. Note that a region can be represented by a *diagonal clock constraint* that involves comparisons of two clocks. If r is a region, then G_r denotes the unique clock constraint such that $r \subseteq \llbracket G_r \rrbracket$. States of time-abstract graphs are pairs of a location and a region. Given a state (l, r) , $Gds_{\mathcal{A}}(l, r) = \{G \mid \ell \xrightarrow{G, A, \mathcal{X}} \ell' \text{ and } \llbracket r \rrbracket \subseteq \llbracket Proj_{I \cup O}(G) \rrbracket\}$ is the set of constraints whose timing part is satisfied by the region r . So, in (l, r) , we only need to change values of input variables in order to satisfy a constraint of $Gds_{\mathcal{A}}(l, r)$.

Definition 5 (Time-abstract graph). *The time-abstract graph (TAG) of a VDTA $\mathcal{A} = \langle L, X, I, O, l^0, G^0, \Delta_{\mathcal{A}} \rangle$ for a constant K is the VDTA $RG_K(\mathcal{A}) = \langle Reg_K(\mathcal{A}), X, I, O, (l^0, r^0), G^0, \Delta_{RG} \rangle$ where*

- $Reg_K(\mathcal{A}) = L \times Reg_K(X)$ is the set of states of $RG_K(\mathcal{A})$
- The initial state is (l^0, r^0) where $r^0 = \{\bar{0}\}$
- The transition relation, $\Delta_{RG} \subseteq Reg_K(\mathcal{A}) \times \mathcal{G}(I, O, X) \times A(O) \times Reg_K(\mathcal{A})$ is such that:

$$\begin{aligned}
 \mathbf{U1} \quad & (\ell, r) \xrightarrow{Proj_X(G) \wedge G_r, A, \mathcal{X}} (\ell', r') \text{ iff } \exists \ell \xrightarrow{G, A, \mathcal{X}} \ell' \text{ in } \mathcal{A} \text{ s.t. } \llbracket r \rrbracket \subseteq \llbracket Proj_{I \cup O}(G) \rrbracket \\
 & \text{and } r' = r[\mathcal{X} \leftarrow 0] \\
 \mathbf{U2} \quad & (\ell, r) \xrightarrow{G' \wedge G_r, Id_O, Id_X} (\ell, r') \text{ with } G' = \neg(\bigvee_{G \in Gds_{\mathcal{A}}(\ell, r)} Proj_X(G)) \text{ and} \\
 & r' = I_Succ(r)
 \end{aligned}$$

In a TAG, the timing information is also encoded in the states. We move from one state to its time-successor whenever the clocks constraint (corresponding to the region of the time-successor) is satisfied and when the input and output constraints of the outgoing urgent transitions are not satisfied. The values of input variables can change when no urgent transition can be fired.

Proposition 1. *For every VDTA \mathcal{A} , for every natural K larger than the largest integer constant in the clock constraints of \mathcal{A} , $ObsTr(RG_K(\mathcal{A})) = ObsTr(\mathcal{A})$.*

We observe that for every natural K larger than the largest integer constant in the clock constraints of \mathcal{A} , for every $\sigma \in ObsTr(\mathcal{A})$, $Out(\mathcal{A} \text{ Safter } \sigma) = Out(RG_K(\mathcal{A}) \text{ Safter } \sigma)$. In the sequel, we shall only consider such K .

Example 3. Figure 2 represents the time-abstract graph of the VDTA in Example 1. In the figure, we have sometimes omitted to represent timing constraints on time-elapsing transitions for clarity reasons

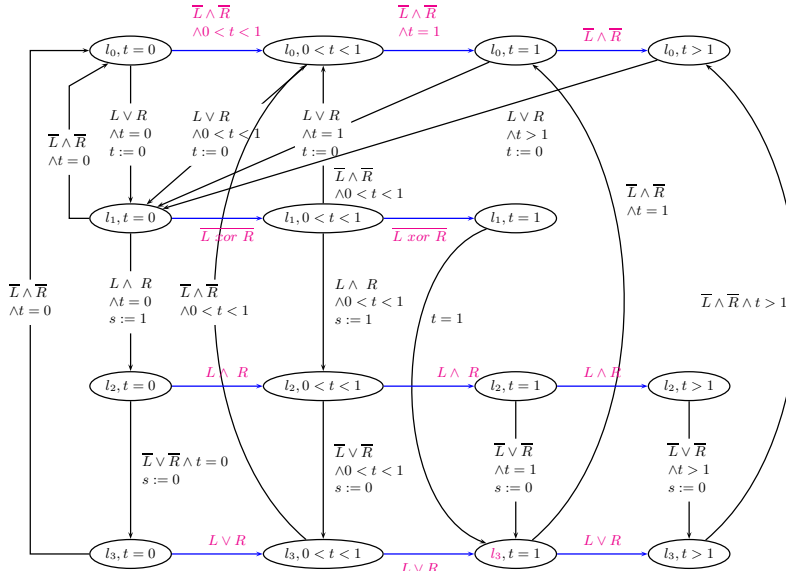


Fig. 2. The TGA of the VDTA of Example 1

3.2 Backward Reachability Analysis

The simple backward control algorithm could work on $\llbracket \mathcal{A} \rrbracket$. It starts in the set of *target* states. Then it computes *predecessors* from which we can reach the target state within 1 step, 2 steps, etc... until an initial state is reached or until the computation terminates. But such a simple algorithm could not terminate

because $\llbracket \mathcal{A} \rrbracket$ is infinite. We consider a symbolic algorithm that works on TAG representations instead of VDTA. We define *abstract predecessors* for configurations of TAG. A configuration of a TAG is a couple of the form (q, G) where q is a state of $RG_K(\mathcal{A})$ and G is a constraint of $\mathcal{G}(I, O)$. We consider *urgent abstract predecessors* ($aPred_u$), *time-elapsing abstract predecessor* ($aPred_\delta$) and *input-update abstract predecessors* ($aPred_e$) defined over as follows:

$$aPred_u(q, G) = \{(q', Proj_X(G') \wedge Proj_{Var(a)}(G) \mid q' \xrightarrow{G', a, Y}_{U_1} q \wedge Proj_{I \cup X}(G')[a] \subseteq Proj_I(G)\}$$

$$aPred_\delta(q, G) = \{(q', Proj_X(G') \wedge G \mid q' \xrightarrow{G', IdO, IdX}_{U_2} q)\}$$

$$aPred_e(q, G) = (q, (\neg \bigvee_{G' \in Gds_{\mathcal{A}}(q)} Proj_X(G')) \wedge Proj_I(G))$$

For a set of configuration Q and $\theta \in \{u, i, \delta\}$, we define the monotonic operators $aPred_\theta(Q, G) = \bigcup_{q \in Q} aPred_\theta(q, G)$. We define the abstract predecessor $aPred(q, G) = aPred_u(q, G) \cup aPred_\delta(q, G) \cup aPred_e(q, G)$. For a set of configurations Q , $aPred(Q) = \bigcup_{q \in Q} aPred(q)$. Finally, $CoReach(Q) = \mu X. Q \cup aPre(X)$ We can show the following proposition that allows to use $CoReach_a$ instead of $Coreach$ during the reachability analysis.

Proposition 2. *Given $q = (l, [x])$ and $G \in \mathcal{G}(I, O)$, $CoReach_a(q, G) = \mu X. (q, G) \cup aPred(X)$ is effectively computable.*

$CoReach_a$ is the least fixpoint of the function $\lambda X. X \cup aPred(X)$ and $aPred : 2^{\mathcal{Q}} \rightarrow 2^{\mathcal{Q}}$ is a monotonic function where $\mathcal{Q} = L \times Reg_K(X) \times \mathcal{C}_M(I, O)$ and $\mathcal{C}_M(I, O)$ denotes the set of constraints on input/outputs the maximal constant occurring in them is lower or equal to M . \mathcal{Q} is finite. Using fixpoint computation results in [?], we get the termination of the computation of $CoReach$. Moreover, we can show that

Proposition 3. *Let $G' \in \mathcal{G}(I, O)$ be a constraints over input and output variables. It holds that*

$$(l, i, o, x) \in Coreach(l', G' \wedge [x']) \text{ iff } ((l, [x]), i, o) \in CoReach_a((l', [x']), G')$$

4 Automatic test generation

4.1 Principle

Conformance testing consists in checking that an implementation exhibits an observable behavior consistent with its specification. We consider conformance testing of critical timed systems modelled with VDTA. We define a conformance relation to ensure that an implementation under test (Imp) conforms to its specification. The main idea of this relation is that all behaviours of the implementation have to be allowed by the specification. Especially:

1. Imp is not allowed to update a variable in a time (too late or too early) when it is not allowed by the specification.
2. Imp is not allowed to omit to change a memory-variable at the time it is required by the specification.

The conformance relation. We assume that Imp and \mathcal{A} are both modeled by compatible VDTA (i.e. they share the same input and output variables). Moreover, we assume that the tester can observe all output variables of the implementation. The tester can only update the input variables of the implementation or let the time elapse. As previously mentioned, the tester can observe the change of the output variables of the implementation only when this one has reached a stable state. As in the ioco theory [15], some states may stay infinitely blocked, but at the moment we do not consider this point in our conformance relation.

Roughly, an implementation conforms with a specification whenever it produces the same outputs as the ones of the specification at the same instants.

Definition 6. *Imp conforms to \mathcal{A} (Imp tuco \mathcal{A}) whenever*

$$\forall \sigma \in ObsTr(\mathcal{A}), Out(Imp \text{ Safter } \sigma) \subseteq Out(\mathcal{A} \text{ Safter } \sigma)$$

where $Out(\cdot)$ gives access to the values of the output variables.

In this relation, we intend to check if the values of output variables of the implementation are correct after any sequence of inputs. These inputs may be of two kinds : assignment of input variables or time elapsing. Since our model permits to fire several transitions in zero time, we decided that the implementation has to reach a stable state before checking correctness of its outputs. For example, if we consider an input followed by several assignments of the same output variable in zero time (e.g. $L := 1$ followed $s := 3$ and $s := 2$), our conformance relation only considers the last value ($s := 2$). Note that it would have been possible to define another relation, similar to *tioco* [7], permitting to consider all kinds of assignments for conformance. The tester who knows the specification plays in the following way:

- either the tester updates the input variables and then observes how the implementation reacts once the implementation is stabilized. In case of non conformance (i.e. if the outputs of the implementation differ from the one of the specification), the tester returns a fail verdict;
- or the tester chooses to let the time elapse for a while; doing so, it observes possible output changes from the implementation. In case such changes are not allowed by the specification at the time a new output observation is performed, the tester returns a fail verdict since the implementation does not conform to the specification;
- or the tester can choose to stop the game and in that case, it returns a pass verdict meaning that, up to this point, no fault occurred.

The tester observes behaviours of the implementation through the values of the output variables in stable states. If their content changes, the tester checks whether the new values are expected by the specification.

4.2 Test purpose

The test selection algorithm we propose is based on the notion of *Test Purpose* (TP). In practice, a test purpose allows to select some behaviors of the specification that we want to test. A test purpose is modeled by a VDTA as follows:

Definition 7. A test purpose TP of a specification $\mathcal{A} = \langle L, X, I, O, l^0, G^0, \Delta_{\mathcal{A}} \rangle$ is a deterministic VDTA $\langle S, X \cup X', I, O, s^0, G^0, \Delta_{TP} \rangle$ such that

- S is a finite set of locations with a special trap location $\text{Accept}_{TP} \in S$, s^0 is the initial location;
- I , O and X are respectively the input, output and clock variables of the specification; TP is thus allowed to observe the configurations of \mathcal{A} ;
- $G^0 \in \mathcal{G}(I, O)$ is the initial condition (the same as the one of \mathcal{A});
- X' is the set of private clocks of TP , with $X' \cap X = \emptyset$
- $\Delta_{TP} \subseteq S \times \mathcal{G}(I, O, X, X') \times \text{Id}_O \times 2^{X'} \times S$ is the transition relation⁴.

Note that TP is non intrusive with respect to the specification. Indeed, according to Definition 7, it does not reset clocks of the specification S and does not assign new values to the output variables. Moreover, we remark that all the test purposes are complete, meaning that whatever is the observation of variables or clocks either a transition is taken or the current location does not change.

Example 4. Some test purposes for the VDTA \mathcal{A} in Fig.1 are presented below.

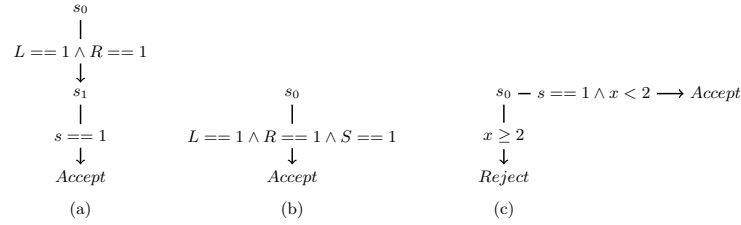


Fig. 3. Test purposes

Test purposes in figures Fig.4(a) and Fig.4(b) observe variables of the VDTA and they only specify which behaviours of the implementation are interesting for the test. The test purpose in Fig.4(c) has its own clock variable x ; it also specifies which behaviour of the implementation should be tested, but also the behaviour of the implementation that should not be tested (from the location *Reject*, the location *Accept*, cannot be reached anymore).

- The test purpose in Fig.4(a) requires that s will be eventually set to 1 after the first time at which L and R are equal to 1. Between the first time in which L and R are simultaneously equal to 1 and the time in which s is set to 1, the test purpose allows, in s_1 , the changing of values L , R and s . The test purpose in Fig.4(b) requires to have s equal to 1 in the same time that L and R are equal to 1.
- With the test purpose described in Fig.4(c), we are only interested in testing behaviours of the implementation for which s is set to 1 at most 2 time units after the beginning of the session. \diamond

⁴ As for the specification, we assume that the guards are given by a boolean combination of elements of $\mathcal{G}(I)$, $\mathcal{G}(O)$, $\mathcal{G}(X)$ and $\mathcal{G}(X')$

4.3 Building the symbolic test case

Given a specification \mathcal{A} and a test purpose TP , we now describe how to derive test cases that target the behaviour of the test purpose while checking for the conformance of the implementation with respect to the specification. It consists in three steps:

Step 1. We first perform the synchronous product between the specification \mathcal{A} and the test purpose in order to characterize in \mathcal{A} the sequences that are accepted by the test purpose TP .

Definition 8 (Synchronous product). *Given $\mathcal{A} = \langle L, X, I, O, l^0, G^0, \Delta_{\mathcal{A}} \rangle$ a specification and a test purpose $TP = \langle S, X' \cup X, I, O, s^0, G^0, \Delta_{TP} \rangle$, the synchronous product of \mathcal{A} and TP is the VDTA $\mathcal{A} \times TP = \langle L \times S, X \cup X', I, O, (l^0, s^0), G^0, \Delta_{\mathcal{A} \times TP} \rangle$ where $\Delta_{\mathcal{A} \times TP}$ is defined by the following rules ($\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$):*

$$\frac{l \xrightarrow{G, A, \mathcal{X}} l' \in L \quad s \in S \quad G_s = \bigwedge_{G' \in G_{TP}(s)} \neg G'}{(l, s) \xrightarrow{G \wedge G_s, A, \mathcal{X}} (l', s)} \quad (\mathcal{R}_1)$$

$$\frac{l \in L \quad s \xrightarrow{G, Id_O, \mathcal{X}'} s' \quad G_l = \bigwedge_{G' \in G_{\mathcal{A}}(l)} \neg G'}{(l, s) \xrightarrow{G_l \wedge G', Id_O, \mathcal{X}'} (l, s')} \quad (\mathcal{R}_2)$$

$$\frac{l \xrightarrow{G, A, \mathcal{X}} l' \quad s \xrightarrow{G', Id_O, \mathcal{X}'} s'}{(l, s) \xrightarrow{G \wedge G', A, \mathcal{X} \cup \mathcal{X}'} (l', s')} \quad (\mathcal{R}_3)$$

Evolutions (transition firing) in the test purpose and the specification depend on clock values and variable values. An urgent transition can be fired in the specification (and not in the test purpose) when the clocks values and the variables values satisfy no constraint on transitions from the current location in the test purpose; this situation is described by the rule \mathcal{R}_1 . Conversely, an urgent transition can be fired in the test purpose (and not in the specification) when no urgent transition is fireable in the specification; this situation is described by the rule \mathcal{R}_2 . They both trigger an urgent transition whenever the values of the variables satisfy the guards of the specification and test purpose transitions. Recall that the test purpose and the specification are deterministic; in consequence when a transition in the specification is fireable, there is at most one fireable transition in the test purpose and reciprocally. This situation is described by the rule \mathcal{R}_3 .

Given a specification \mathcal{A} and a test purpose TP , we denote $\mathcal{A}_{TP} = \mathcal{A} \times TP$. Note that \mathcal{A} is the master of this composition in the sense that it is the only one able to change the values of the input and output variables. Due to rule \mathcal{R}_1 and \mathcal{R}_3 , there is no imposed restriction with respect to the behaviour of \mathcal{A} . In other words, the definition of $\mathcal{A}_{TP} = \mathcal{A} \times TP$ does restrict the constraints that allow, in \mathcal{A} , to execute some output updates. Since input-update and delay can only be performed when the constraints that allow an output update are unsatisfiable, we can show that $Tr(\mathcal{A}) = Tr(\mathcal{A}_{TP})$ and $ObsTr(\mathcal{A}) = ObsTr(\mathcal{A}_{TP})$. In the sequel, we shall denote *Accept* the set of states of \mathcal{A}_{TP} of the form $(l, Accept_{TP})$.

Remark 2. \mathcal{A}_{TP} has at most $|L| \times |S|$ locations and $\sum_{(l,s) \in L \times S} (|G_{\mathcal{A}}(l)| \times |G_{TP}(s)|) + 2$ transitions.

Step 2: Test Selection. From \mathcal{A}_{TP} , we build the corresponding region graph to abstract away the time: $RG(\mathcal{A}_{TP}) = \langle Q, X, I, O, q^0, G^0, \Delta_{RG} \rangle$. We denote by *Pass* the set of locations of the form $(Accept, r) \in Q$. From the test generation point of view, our aim is to generate test cases that allow to reach the *Accept* location. We thus consider the set of constrained configurations $\mathcal{Q}_{Pass} = \{(q, true) \mid q \in Pass\}$ and we compute the set of coreachable corresponding constraints. It is given by $CoReach_a(\mathcal{Q}_{Pass}) = \cup_{P \in \mathcal{Q}_{Pass}} CoReach_a(P)$. Intuitively, during the computation of $CoReach_a(\mathcal{Q}_{Pass})$, if we encounter a symbolic state (q, G) of $RG_K(\mathcal{A}_{TP})$ with G as constraint on the input/output variables, then there will exist a path giving a way to move from q to *Pass* by letting the time elapse or by changing inputs conveniently in encountered locations along the path. Note that when computing $CoReach_a(q, G)$ for a given symbolic state (q, G) of $RG_K(\mathcal{A}_{TP})$, we can tag visited locations q of $RG_K(\mathcal{A}_{TP})$ with adequate constraints on the input/output variables. We call a *symbolic test case*, a path from the initial location of $RG_K(\mathcal{A}_{TP})$ to a *Pass* location.

4.4 Test case execution

We assume that we have selected a symbolic test case that is a path in $RG_K(\mathcal{A}_{TP})$ that ends in a location of the form $(Accept, r)$ of *Pass* for some region r . Let $TC = p_0.p_1 \dots p_n$ be such a symbolic test case where in each position $p_k = ((l_k, r_k), I_k)$ with $k = 1..n$, l_k denotes a location of the specification, with $l_n = Accept_{TP}$, r_k denotes a region and I_k denotes a constraints (invariant) over input/output variables computed w.r.t. $CoReach_a(\mathcal{Q}_{pass})$. We provide an on-the-fly testing algorithm for TC . The algorithm works as follows:

Let j be the position in the symbolic test case that contains the current stable state $st_j = ((l_j, r_j), i_j, o_j, x_j)$ and $Otr_j = \sigma_1 \dots \sigma_j \in (A(I) \cup \mathbb{R}_+)^*$ be the sequence played on the implementation so far.

Begin loop

- (a) If $(l_j, r_j) \in Pass$ and $Out(Impl \text{ Safter } Otr_j) = o_j$ then exit **loop** and return the verdict “pass”.
- (b) Choose either to delay, or to perform an input-update

The decision is to perform an input-update.

- i. Select an assignment A_j such that $(i_j[A_j], o_j, x_j)$ satisfies the constraint on the transition, in TC , starting from $((l_j, s_j), r_j)$ and compute $st_{j+1} = st_j \text{ Safter } A_j$
- ii. update the inputs of the implementation according to A_j
- iii. If $Out(Impl \text{ Safter } Otr_j.A_j) \neq o_{j+1}$ exit **loop** and return the verdict “fail”; else st_j becomes st_{j+1} .

The decision is to delay

- i. Pickup $\delta \in \mathbb{R}_+$ such that $(i_j, o_j, x_j + \delta)$ satisfies the constraint on the transition, in TC , starting from (l_j, r_j) .

- ii. If an output update occurs on Imp within $\delta' \leq \delta$ time units then compute $st_{j+1} = st_j$ Safter δ'
- iii. If $Out(Imp \text{ Safter } Otr_j.\delta') \neq o_{j+1}$ exit **loop** and return the verdict “fail”; else st_j becomes st_{j+1} .

End loop

5 Conclusion

In this work, we have been interested in the automatic test generation for data-flow systems. In order to model such systems, we have presented the Variable Driven Timed Automata model (VDTA) : a variant of timed automata with continuous variables partitioned into two sets : input variables and output ones. Input variables permit to control the system and output variables are considered as the observable outputs. Transitions are urgent and it is possible to fire several transitions in zero time in a synchronous way. We have also proposed a new conformance relation adapted to our model, and a test generation method with a selection based on a test purposes.

An interesting extension of this work would be to handle blocking states in the conformance relation. Besides, we also intend to consider assignments of variables with operations depending on other variable values (e.g. $x := y + 3$). Since reachability and coreachability problems become undecidable, we propose to use abstract interpretation and approximation techniques.

Acknowledgment

This work has been supported by the ANR⁵ Testec Project. We also would like to thank Nathalie Bertrand and Thierry Jéron for their help.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. Roberto Barbuti and Luca Tesi. Timed automata with urgent transitions. *Acta Inf.*, 40(5):317–347, 2004.
3. L. Du Bousquet, F. Ouabdesselam, J.-L. Richier, and N. Zuanon. Lutess: A specification-driven testing environment for synchronous software. In *International Conference on Software Engineering*, pages 267–276, 1999.
4. Rachel Cardell-Oliver. Conformance testing of real-time systems with timed automata specifications. *Formal Aspects of Computing Journal*, 12(5):350–371, 2000.
5. Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John Plaice. Lustre: A declarative language for programming synchronous systems. In *Symposium on Principles of Programming Languages, POPL 1987: Munich, Germany*, pages 178–188, 1987.

⁵ Agence Nationale de la Recherche

6. Abdeslam EnNouaary, Rachida Dssouli, and Ferhat Khendek. Timed wp-method: Testing real-time systems. *IEEE Transactions on Software Engineering (TSE)*, 28(11):1023–1038, 2002.
7. Moez Krichen and Stavros 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*, pages 109–126. Springer, 2004.
8. Bruno Marre and Agnes Arnould. Test sequences generation from lustre descriptions: Gatel. In *ASE '00: Proceedings of the 15th IEEE international conference on Automated software engineering*, 2000.
9. Marius Mikucionis, Kim Guldstrand Larsen, and Brian 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*, pages 396–397. IEEE Computer Society, 2004.
10. Houda Bel Mokadem, Béatrice Bérard, Patricia Bouyer, and François 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*, pages 110–124, 2005.
11. Manuel Núñez and Ismael 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*, pages 103–117. Springer, 2005.
12. P. Raymond, X. Nicollin, N. Halbwachs, and D. Waber. Automatic testing of reactive systems, madrid, spain. In *Proceedings of the 1998 IEEE Real-Time Systems Symposium, RTSS'98*, pages 200–209. IEEE Computer Society Press, December 1998.
13. Besnik Seljimi and Ioannis Parissis. Using clp to automatically generate test sequences for synchronous programs with numeric inputs and outputs. In *ISSRE '06: Proceedings of the 17th International Symposium on Software Reliability Engineering*, pages 105–116, Washington, DC, USA, 2006. IEEE Computer Society.
14. J. Springintveld, F.W. Vaandrager, and P. R. D'Argenio. Timed Testing Automata. *Theoretical Computer Science*, 254(254):225–257, 2001.
15. J. Tretmans. Test generation with inputs, outputs, and repetitive quiescence. 17:103–120, 1996.

6 Appendix: Proof of Proposition 1

Proposition 1 is a corollary of Lemma 7 and Lemma 6

Proposition 1 Let \mathcal{A} be a VDTA, for every K greater or equal to the maximal constants clocks in \mathcal{A} are compared to, we have that:

$$ObsTr(RG_K(\mathcal{A})) = ObsTr(\mathcal{A})$$

Property 1 (Time Additivity in Obs(\mathcal{A})). Let $\delta', \delta'' \in \mathbb{R}_+$. It holds that:

$$(l, i, o, x) \xrightarrow{\delta'} (l', i', o', x') \text{ and } (l', i', o', x') \xrightarrow{\delta''} (l'', i'', o'', x'') \text{ if and only if } (l, i, o, x) \xrightarrow{\delta' + \delta''} (l'', i'', o'', x'')$$

Lemma 1. (l, i, o, x) is stable if and only if $((l, [x]), i, o, x)$ is stable.

Proof. (if) Part: Assume that (l, i, o, x) is stable, then by definition, for every $A \in A(O)$, $(l, i, o, x) \not\stackrel{A}{\rightarrow}$, or equivalently for all $G \in G_{\mathcal{A}}(l)$, for every $tr_G = (l \xrightarrow{G, A, \mathcal{X}} l')$ either (1) $(i, o) \not\models Proj_X(G)$ or (2) $x \not\models Proj_{IUO}(G)$. Consider now the location $(l, [x])$, urgent transitions starting from this location are of the form:

- $t = ((l, [x]) \xrightarrow{G \wedge G_{[x]}, A, \mathcal{X}}_{U_1} (l', [x'])),$ with $[x'] = [x][\mathcal{X} \leftarrow 0]$. For this transition, tr_G can not be triggered in \mathcal{A} because of (1) or (2). If $(i, o) \not\models Proj_X(G)$, then obviously t can not be triggered. If $x \not\models Proj_{IUO}(G)$, then as $G_{[x]} = Proj_{IUO}(G)$, it is not possible to trigger t .
- $t = ((l, [x]) \xrightarrow{G' \wedge G_{[x]}, Id_O, \emptyset}_{U_2} (l', [x'])),$ with $[x'] = I_{\perp} Succ([x])$ and $G' = G_{[x']} \wedge (\neg \bigvee_{G \in Gds_{\mathcal{A}}(l, [x])} Proj_X(G))$. As $\llbracket G_{[x']} \wedge G_{[x]} \rrbracket = \emptyset$, we get that $x \not\models G_{[x']}$ and t can not be triggered.

(only if) Part: Assume that $((l, [x]), i, o, x)$ is stable, then for every $a \in A(O)$, it holds that $((l, [x]), i, o, x) \not\stackrel{A}{\rightarrow}$; or equivalently, for every $tr_G = ((l, [x]) \xrightarrow{G, A, \mathcal{X}} (l', [x'])),$ either $(i, o) \not\models Proj_X(G)$ or $x \not\models Proj_{IUO}(G)$. We show that no urgent transition (of type **T1**) can be fired from (l, i, o, x) . By contradiction, if a transition of type **T1** can be fired from (l, i, o, x) , then there would exist a transition $tr = l \xrightarrow{G', A', \mathcal{X}'} l'$ such that $(i, o) \models Proj_X(G')$ and $x \models Proj_{IUO}(G')$. Using a dual argument to the proof of the **if** part, we get that there is a transition of type **T1** from $((l, [x]), i, o, x)$, meaning that the latter state is not stable. This is a contradiction.

Lemma 2. Given $A \in A(O) \setminus \{Id_O\}$,

$$(l, i, o, x) \xrightarrow{A} (l', i', o', x') \text{ iff } ((l, [x]), i, o, x) \xrightarrow{A} ((l', [x']), i', o', x')$$

Proof. First note that if $(l, i, o, x) \xrightarrow{A} (l', i', o', x')$ then (l, i, o, x) is not stable and, according to Lemma 1, $((l, [x]), i, o, x)$ is not stable.

(if) Part: Consider $A \in A(O) \setminus \{Id_O\}$. Assume that $(l, i, o, x) \xrightarrow{A} (l', i', o', x')$, then there exists $tr = (l \xrightarrow{G, A, \mathcal{X}} l')$ such that $(i, o) \models Proj_X(G)$, $x \models Proj_{IUO}(G)$, $i' = i$, $o' = o[A]$ and $x' = x[\mathcal{X} \leftarrow 0]$. By definition, there exists in $RG_K(\mathcal{A})$ a transition $(l, [x]) \xrightarrow{G', A, \mathcal{X}'} (l', [x][\mathcal{X} \leftarrow 0])$ with $G' = Proj_X G \wedge G_{[x]}$. As $G_{[x]}$ is the unique atomic clock constraint that contains $[x]$, we have that $x \models G_{[x]}$ and then the transition $((l, [x]), i, o, x) \xrightarrow{A} ((l', [x']), i', o', x')$ exists.

(only if) Part: Conversely, assume that $((l, [x]), i, o, x) \xrightarrow{A} ((l', [x']), i', o', x')$. Then there exists in $RG_K(\mathcal{A})$ a transition $tr = (((l, [x]), i, o, x) \xrightarrow{G, A, \mathcal{X}} ((l', [x']), i', o', x'))$ such that $(i, o) \models Proj_X(G)$, $x \models Proj_{IUO}(G)$, $i' = i$, $o' = o[A]$ and $[x'] = [x][\mathcal{X} \leftarrow 0]$. But tr is constructed using some transition $l \xrightarrow{G', A, \mathcal{X}'} l'$ such that

$Proj_X(G) = Proj_X(G')$, $Proj_{I\cup O}(G) = G_{[x]}$ and $\llbracket Proj_{I\cup O}(G) \rrbracket \subseteq \llbracket Proj_{I\cup O}(G') \rrbracket$. As $(i, o) \models Proj_X(G)$, $x \models Proj_{I\cup O}(G)$ we have that $x \models Proj_{I\cup O}(G')$ and then the transition $(l, i, o, x) \xrightarrow{A} (l', i', o', x')$ exists.

Remark 3. For every state q of $RG_K(\mathcal{A})$, it holds that $\bigvee_{G \in RG_K(\mathcal{A})(q)} Proj_X(G)$ is a tautology.

Lemma 3. *Let $\delta \in \mathbb{R}_+$. Then,*

$$((l, [x]), i, o, x) \xrightarrow{\delta} ((l', r), i, o, x') \text{ implies } (l, i, o, x) \xrightarrow{\delta} (l, i, o, x')$$

Proof. Assume that $((l, [x]), i, o, x) \xrightarrow{\delta} ((l', r), i, o, x')$. Then, by definition $x' = x + \delta$ and for every transition $((l, [x]) \xrightarrow{G, A, \mathcal{X}} (l', r')$, for every $\delta' < \delta$, $Proj_{I\cup O}(G) = G_{[x]}$ and either $(i, o) \not\models Proj_X G$ or $x + \delta' \not\models Proj_{I\cup O} G$. Assume now that $(l, i, o, x) \not\xrightarrow{\delta}$, then there exists $\delta' < \delta$ and $A \in A(0)$ such that $(l, i, o, x) \xrightarrow{\delta'} (l, i, o, x + \delta')$ and $(l, i, o, x + \delta') \xrightarrow{A} (l', i, o', x'')$. Thus there exists $tr = l \xrightarrow{G', A, \mathcal{X}'} l'$ such that $(i, o) \models Proj_X(G')$, $x + \delta' \models Proj_{I\cup O}(G')$, $o' = o[A]$, and $x'' = x[\mathcal{X}' \leftarrow 0]$. Note that by definition, $x + \delta' \in [x]$. Moreover, as tr is a transition of \mathcal{A} , there exists a transition $(l, [x]) \xrightarrow{Proj_X(G') \wedge G_{[x]}, A, \mathcal{X}'} (l', [x''])$ in $RG_K(\mathcal{A})$. Clearly, $(i, o) \models Proj_X(G')$ and $x + \delta' \models G_{[x]}$ involving that there is a urgent transition of type **T1** from $((l, [x]), i, o, x)$ when we delay for δ' . As urgent transition of type **T1** are taken prior to delay transition of type **T3**, we get $((l, [x]), i, o, x) \not\xrightarrow{\delta}$ and this is a contradiction with the hypothesis.

Lemma 4. *Let x and x' such that $[x'] = I_Succ([x])$ and let $\delta \in \mathbb{R}_+$. $(l, i, o, x) \xrightarrow{\delta} (l, i, o, x')$ if and only if $((l, [x]), i, o, x) \xrightarrow{\delta, Id_O} ((l', [x']), i, o, x')$*

Proof. (if) Part: Assume that $(l, i, o, x) \xrightarrow{\delta} (l, i, o, x')$, then $x' = x + \delta$ and by definition, for every $l \xrightarrow{G, A, \mathcal{X}} l'$ and for every $\delta' < \delta$ either $(i, o) \not\models Proj_X(G)$ or $x + \delta' \not\models Proj_{I\cup O}(G)$. By definition, there exists in $RG_K(\mathcal{A})$ a transition $(l, [x]) \xrightarrow{G' \wedge G_{[x]}, Id_O, \emptyset} (l', [x'])$ such that $\llbracket G_{[x']} \rrbracket \subseteq \llbracket Proj_{I\cup O}(G') \rrbracket$ and for every $G'' \in Gds_{\mathcal{A}}(l, [x])$, it holds that $\llbracket Proj_{I\cup O}(G') \rrbracket \cap \llbracket Proj_{I\cup O}(G'') \rrbracket = \emptyset$. As $x \not\models G_{[x]}$, and $(i, o) \models Proj_X(G')$, a delay transition of amount δ can be fired from $((l, [x]), i, o, x)$, meaning that the transition $((l, [x]), i, o, x) \xrightarrow{\delta} ((l, [x]), i, o, x + \delta)$ exists. Additionnaly, as $x' = x + \delta$ and $x' \models G_{[x']}$, we get the existence of the urgent transition (of type **T1**) $((l, [x]), i, o, x + \delta) \xrightarrow{Id_O} ((l, [x']), i, o, x')$ with $x' = x + \delta$. In conclusion we have shown that $((l, [x]), i, o, x) \xrightarrow{\delta} ((l, [x]), i, o, x') \xrightarrow{Id_O} ((l, [x']), i, o, x')$.

(only if) part: if $((l, [x]), i, o, x) \xrightarrow{\delta, Id_O} ((l', [x']), i, o, x')$ then we have $((l, [x]), i, o, x) \xrightarrow{\delta} ((l, [x]), i, o, x') \xrightarrow{Id_O} ((l, [x']), i, o, x')$ with $x' = x + \delta$. Using lemma 3, we get that $(l, i, o, x) \xrightarrow{\delta} (l, i, o, x')$.

Corollary 1. Let $\delta \in \mathbb{R}_+$. Then, $(l, i, o, x) \xrightarrow{\delta} (l', i', o', x')$ if and only if it exists in $\llbracket RG_K(\mathcal{A}) \rrbracket$ a sequence of transitions

$$((l, [x]), i, o, x) \xrightarrow{\delta_1.Id_O} ((l'_1, [x'_1]), i'_1, o'_1, x'_1) \xrightarrow{\delta_2.Id_O} \dots \xrightarrow{\delta_n.Id_O} ((l', [x']), i', o', x')$$

with $\delta = \sum_{i=1}^n \delta_i$.

Proof. The proof relies on Lemma 4

Lemma 5. Let $A \in A(I)$. Then, $(l, i, o, x) \xrightarrow{A} (l', i', o, x)$ if and only if $((l, [x]), i, o, x) \xrightarrow{A} ((l', [x']), i', o, x)$

Proof. (if) Part: Assume that $(l, i, o, x) \xrightarrow{A} (l', i', o, x)$ with $i' = i[A]$ can be triggered in $\llbracket \mathcal{A} \rrbracket$, then no transition of type **T1** can be triggered from (l, i, o, x) . Thus, for every $t_G = (l \xrightarrow{G, A, \mathcal{X}} l')$, it holds that $(i, o) \not\models Proj_X(G)$ or $x \not\models Proj_{I \cup O}(G)$. For contradiction, assume that $((l, [x]), i, o, x) \not\xrightarrow{A}$. In that case, there exists in $RG_K(\mathcal{A})$ an urgent transition $tr = (l, [x]) \xrightarrow{G', A', \mathcal{X}'} (l'', r)$ such that $(i, o) \models Proj_X G'$, $Proj_{I \cup O}(G') = G_{[x]}$ and $x \models Proj_{I \cup O}(G')$. In turns, this implies the existence of a transition $l \xrightarrow{G'', A', \mathcal{X}''} l''$ with $Proj_X(G'') = Proj_X G'$ and $\llbracket Proj_{I \cup O}(G'') \rrbracket \subseteq Proj_{I \cup O}(G')$. It entails that there exists a transition $(l, i, o, x) \xrightarrow{A'} (l'', i, o[A'], x[\mathcal{X}' \leftarrow 0])$ in $\llbracket \mathcal{A} \rrbracket$ which discards the existence of the input update transition $(l, i, o, x) \xrightarrow{A} (l', i', o', x')$. So the contradiction.

(only if) Part: Similar to the **(if) part**

Lemma 6. Let (l, i, o, x) and (l', i', o', x') be two stable states, then

$$(l, i, o, x) \xrightarrow{A_I} (l', i', o', x') \text{ if and only if } ((l, [x]), i, o, x) \xrightarrow{A_I} ((l', [x']), i', o', x')$$

Proof. (if) Part: Assume that $(l, i, o, x) \xrightarrow{A_I} (l', i', o', x')$, then there exist a set $\{A_k \mid k = 1..n\} \subseteq A(O) \cup \{0\}$ and a sequence of transitions in $\llbracket \mathcal{A} \rrbracket$ such that

$$(l, i, o, x) \xrightarrow{A_I} (l_0, i_0, o_0, x_0) \xrightarrow{A_1} \dots \xrightarrow{A_n} (l_n, i_n, o_n, x_n)$$

W.L.O.G, assume that every A_k belongs $\subseteq A(O)$; then there is a set of transitions $\{l_{k-1} \xrightarrow{G_k, A_k, \mathcal{X}_k} l_k \mid k = 1..n\}$ such that:

- $i_0 = i[A_I]$, $o_0 = o$, $x_0 = x$ and for every $1 \leq k \leq n$, $i_k = i_o$
- for every $0 \leq k \leq n-1$, $(i_k, o_k, x_k) \models G_k$
- for every $1 \leq k \leq n$, $o_k = o_{k-1}[A_k]$ and $x_k = x_{k-1}[\mathcal{X}_k \leftarrow 0]$
- $(l_n, i_n, o_n, x_n) = (l', i', o', x')$

Applying Lemma 5 and Lemma 2 we get the existence of the sequence of transitions $((l, [x]), i, o, x) \xrightarrow{A_I} (l_0, [x_0], i_0, o_0, x_0) \xrightarrow{A_1} \dots \xrightarrow{A_n} (l_n, [x_n], i_n, o_n, x_n)$ where $(l_n, [x_n], i_n, o_n, x_n) = (l', [x']), i', o', x'$. Moreover, according to Lemma 1, both $(l, [x]), i, o, x)$ and $(l', [x']), i', o', x')$ are stable states and finally,

$$(l, [x]), i, o, x) \xrightarrow{A_I} (l', [x']), i', o', x')$$

(only if) Part: Similar to the **(if) Part**.

Lemma 7. *It holds that*

$$(l, i, o, x) \xrightarrow{\delta} (l', i', o', x') \text{ if and only if } ((l, [x]), i, o, x) \xrightarrow{\delta} ((l', [x']), i', o', x').$$

Proof. This is a direct consequence of Lemmas 4 and 2 using a construction similar to the one of the previous lemma.

7 Appendix: Proof of Proposition 3

We give a proof of Proposition 3 that establishes the link between *CoReach* and *CoReach_a*. The proposition show that we can use *CoReach_a* instead of *CoReach* during the reachability analysis.

Proposition 3 Let $G' \in \mathcal{G}(I, O)$ be a constraints over input and outputs variables. It holds that

$$(l, i, o, x) \in \text{CoReach}(l', G' \wedge [x']) \text{ iff } ((l, [x]), i, o) \in \text{CoReach}_a((l', [x']), G')$$

Note that *CoReach* and *CoReach_a* are least fixpoint of the monotonic operators *Pred* and *aPred*. Then using Lemma 8, Lemma 9, Lemma 10 all presented below, we will show in Lemma 11 that *aPred* can be used instead of *Pred* when computing predecessors of states. First of all let us make more precise the computation of predecessors of states of VDTA.

7.1 Predecessors

The semantics $\llbracket \mathcal{A} \rrbracket$ of a VDTA \mathcal{A} is a $\Sigma - LTS$ with three kind of transitions. Then, for each state $s = (l, i, o, x)$ we consider three sorts of predecessors:

- the urgent predecessor $Pred_u$ is defined by

$$Pred_u(l', i', o', x') = \{ (l, i, o, x) \mid \exists a \in A(O) \text{ s.t. } (l, i, o, x) \xrightarrow{A}_{T_1} (l', i', o', x') \}$$

- the input-update predecessor $Pred_e$ is defined by

$$Pred_e(l', i', o', x') = \{ (l, i, o, x) \mid \exists a \in A(I)(l, i, o, x) \xrightarrow{A}_{T_2} (l', i', o', x') \}$$

Note that by definition $(l, i, o, x) \in Pred_e(l', i', o', x')$ iff $l = l'$, $o = o'$, $\exists a \in A(I)$ s.t. $i' = i[A]$ and $(l, i, o, x) \not\xrightarrow{T_1}$

- the time elapsing predecessor $Pred_\delta$ is defined by

$$Pred_\delta(l', i', o', x') = \{ (l, i, o, x) \mid \exists \delta \in \mathbb{R}_+ \text{ s.t. } (l, i, o, x) \xrightarrow{\delta}_{T_3} (l', i', o', x') \}$$

Note that by definition $(l, i, o, x) \in Pred_\delta(l', i', o', x')$ iff $l = l'$, $i = i'$, $o = o'$, and $\forall 0 \leq \delta' < \delta$, $(l, i, o, x + \delta') \not\xrightarrow{T_1}$

Finally we define

$$Pred(l, i, o, x) = Pred_u(l, i, o, x) \cup Pred_e(l, i, o, x) \cup Pred_\delta(l, i, o, x)$$

For $\theta \in \{u, i, \delta\}$, and a set of state Q , we define $Pred_\theta(Q) = \bigcup_{s \in Q} Pred_\theta(s)$.

Note that $Pred_\theta : 2^S \rightarrow 2^S$ is monotonic and then $Pred : 2^S \rightarrow 2^S$ defined by $Pred(Q) = \bigcup_{s \in Q} Pred(s)$ is also monotonic. For a constraint $G \in \mathcal{G}(I, O, X)$ and a location l , the configuration (l, G) denotes a set of states and it is defined by $(l, G) = \{(l, i, o, x) \mid (i, o, x) \models G\}$.

7.2 Relation between $Pred$ and $aPred$

Let $G' \in \mathcal{G}(I, O)$ be a constraints over input and outputs variables. We provide proofs to following lemmas. The proofs use results in Lemma 1, Lemma 2, and Lemma 5.

Lemma 8.

$(l, i, o, x) \in Pred_u(l', G' \wedge [x'])$ if and only if $((l, [x]), i, o) \in aPred_u(l', [x']), G'$.

Proof. (if) Part: We assume that $(l, i, o, x) \in Pred_u(l', G' \wedge [x'])$ and we show $((l, [x]), i, o) \in aPred_u(l', [x']), G'$. If $(l, i, o, x) \in Pred_u(l', G' \wedge [x'])$ then there exists $A \in A(O)$ and $(l', i', o', x') \in (l', G' \wedge [x'])$ such that $(l, i, o, x) \xrightarrow{A} (l', i', o', x')$ and by Lemma 2 we have that $((l, [x]), i, o) \xrightarrow{A} ((l', [x']), i', o')$. It remains to show that there is a symbolic state in $Z \in aPred(l', [x']), G'$ such that $((l, [x]), i, o) \in Z$. By definition,

$$aPred_u(l', [x']), G' = \{(q, Proj_X(G') \wedge Proj_{Var(a)}(G) \mid q \xrightarrow{G, A, Y}_{U_1} (l', [x']) \wedge Proj_{I \cup X}(G)[A] \subseteq Proj_I(G')\}$$

If $(l, i, o, x) \xrightarrow{A} (l', i', o', x')$, then there exists $tr = l \xrightarrow{G_l, A, \mathcal{X}} l'$ such that $(i, o) \models Proj_X(G_l)$, $x \models Proj_{I \cup O}(G_l)$, $i' = i$, $o' = o[A]$, $x' = x[\mathcal{X} \leftarrow 0]$. As tr exists, we have that $((l, [x]) \xrightarrow{Proj_X(G_l) \wedge G_{[x], A, \mathcal{X}}} (l', [x']$). Let $Z = ((l, [x]), Proj_X(G_l) \wedge Proj_{Var(a)}(G'))$, we have that $Z \in aPred(l', [x']), G'$. It is not difficult to convince that $((l, [x]), i, o) \in Z$. As we already have that $(i, o) \models Proj_X(G_l)$, it remains to establish that $(i, o) \models Proj_{Var(a)}(G')$. This is true as, by hypothesis we have that $(i', o') \models G'$, $(i, o) \models Proj_X(G_l)$, $Proj_{I \cup X}(G_l)[A] \subseteq Proj_I(G')$, $i' = i$ and $o' = o[A]$.

(only if) Part: Similar to the **(if) Part**.

Lemma 9.

$(l, i, o, x) \in Pred_e(l', G' \wedge [x'])$ if and only if $((l, [x]), i, o) \in aPred_e(l', [x']), G'$

Proof. (if) Part: We assume that $(l, i, o, x) \in Pred_e(l', G' \wedge [x'])$ and we show that $((l, [x]), i, o) \in aPred_e(l', [x']), G'$. By definition $(l, i, o, x) \in Pred_e(l', G' \wedge [x'])$ implies that there exists $A \in A(I)$, $(l', i', o', x') \in (l', G' \wedge [x'])$ such that $(l, i, o, x) \xrightarrow{A} (l', i', o', x')$ with $l' = l$, $i' = i[A]$, $o' = o$, $x' = x$ and by Lemma 5 we have that $((l, [x]), i, o) \xrightarrow{A} ((l, [x]), i', o)$. Now we show that $((l, [x]), i, o) \in aPred_e(l', [x']), G'$ where

$$aPred_e(l', [x]), G' = \left((l, [x]), \left(\neg \bigvee_{G \in Gds_{\mathcal{A}}(l, [x])} proj_X(G) \right) \wedge Proj_I(G') \right)$$

and it is equivalent to show that

$$(i, o) \models \left(\neg \bigvee_{G \in Gds_{\mathcal{A}}(l, [x])} proj_X(G) \right) \wedge Proj_I(G')$$

Obviously, we have that $(i, o) \models \text{Proj}_I(G')$ because $(i', o') \models G'$ and $o' = o$. Now it remains to show that $(i, o) \models (\neg \bigvee_{G \in \text{Gds}_{\mathcal{A}}(l, [x])} \text{proj}_X(G))$. Observe that for every G such that a transition $(l, [x]) \xrightarrow{G, A, \emptyset}_{U_1} (l'', r'')$ exists in $RG_K(\mathcal{A})$, it holds that $[x] \subseteq \text{Proj}_{I \cup O}(G)$ and there is a symbolic transition in \mathcal{A} $tr = l \xrightarrow{G_l, A, Y} l'$ such that $\text{Proj}_X(G_l) = \text{Proj}_X(G)$. Since $(l, i, o, x) \xrightarrow{A} (l', i', o', x')$, then for every $l \xrightarrow{G_l, A, X} l'$ it is true that $(i, o) \not\models \text{Proj}_X(G_l)$ or $x \not\models \text{Proj}_{I \cup O}(G_l)$. In particular, for every constraint G_l in the set $\mathcal{C} = \{G_l \mid l \xrightarrow{G_l, A, X} l' \wedge x \models \text{Proj}_{I \cup O}(G_l)\}$ we will have that $(i, o) \not\models \text{Proj}_X(G_l)$ or equivalently $(i, o) \not\models \bigwedge_{G_l \in \mathcal{C}} \text{Proj}_X(G_l)$ that in turn is equivalent to $(i, o) \models \neg \bigvee_{G_l \in \mathcal{C}} \text{Proj}_X(G_l)$. Note that $\mathcal{C} = \text{Gds}_{\mathcal{A}}(l, [x])$ and then we get that $(i, o) \models (\neg \bigvee_{G \in \text{Gds}_{\mathcal{A}}(l, [x])} \text{proj}_X(G))$.
(only if) Part: Similar to the **(if) Part**.

Lemma 10.

$(l, i, o, x) \in \text{Pred}_{\delta}(l', G \wedge [x'])$ if and only if there exists $k > 1$ such that $((l, [x]), i, o) \in a\text{Pred}_{\delta}^k((l', [x']), G')$

Proof. (if) Part: We assume that $(l, i, o, x) \in \text{Pred}_{\delta}(l', G' \wedge [x'])$ and we show that $((l, [x]), i, o) \in a\text{Pred}_{\delta}^k((l', [x']), G')$. By definition $(l, i, o, x) \in \text{Pred}_{\delta}(l', G' \wedge [x'])$ implies that there exists $\delta \in \mathbb{R}_+$, $(l', i', o', x') \in (l', G' \wedge [x'])$ such that $(l, i, o, x) \xrightarrow{\delta} (l', i', o', x')$ with $l' = l$, $i' = i$, $o' = o$, $x' = x + \delta$. Let k denote the distance (or the number of distinct regions) from $[x]$ to $[x']$. Then there are $\delta_1, \delta_2, \dots, \delta_k$ in \mathbb{R}_+ and regions $r_0, r_1, r_2, \dots, r_k$ such that $\delta = \sum_{l=1}^k \delta_l$, $r_0 = [x]$, $r_k = [x']$, $r_j = [x + \sum_{l=1}^j \delta_l]$ and $r_j = I\text{-Succ}(r_{j-1})$ with $j \in \{1..k\}$. Now we show that $((l, [x]), i, o) \in a\text{Pred}_{\delta}^k((l, [x]), G')$ where

$$a\text{Pred}_{\delta}((l, [x']), G') = \{((l, [x']), G')\} \cup \{(q', \text{Proj}_X(G_q) \wedge G') \mid q' \xrightarrow{G_q, Id_O, \emptyset}_{U_2} (l, [x])\}$$

We consider the two cases:

- If $k = 1$ then $[x] = [x']$ and we need to show that $(l, [x]), i, o) \in ((l, [x']), G')$ or equivalently we show that $(i, o) \models G'$. Because $(l, i, o, x) \xrightarrow{\delta} (l', i', o', x')$ with $l' = l$, $i' = i$, $o' = o$ and $[x] = [x']$ we have that $((l, [x]), i, o) \xrightarrow{\delta} ((l', [x']), i', o')$ meaning that $((l, [x]), i, o) \in a\text{Pred}_{\delta}^1((l', [x']), i', o')$ and by hypothesis $(i', o') \models G'$ and $i' = i$, $o' = o$.
- If $k > 1$ then by Lemma 4 there exists a sequence of transitions

$$((l, r_0), i_0, o_0, x_0) \xrightarrow{\delta_1, Id_O} ((l, r_1), i_1, o_1, x_1) \xrightarrow{\delta_2, Id_O} \dots \xrightarrow{\delta_k, Id_O} ((l, r_k), i_k, o_k, x_k)$$

and that sequence is such that $x_0 = x$, $x_k = x'$, $x_l = x_{l-1} + \delta_l = i_{j-1} = i_j = i = i'$ and $o_{j-1} = o_j = o = o'$ for every $j \in \{1..k\}$. This is because the time-elapsing operation does not change inputs and outputs. As $(i', o') \models G'$ we get that $(i_j, o_j) \models G'$ for every $j \in \{1..k\}$. Additionnaly each transition $((l, r_{j-1}), i_{j-1}, o_{j-1}, x_{j-1}) \xrightarrow{\delta_j, Id_O} ((l, r_j), i_j, o_j, x_j)$ implies that $(i_j, o_j) \models$

$Proj_X(G_l)$ where G_j is the constraint on the transition $(l, r_{j-1}) \xrightarrow{G_j, Id_O} (l, r_j)$ that must exist.

(only if) Part: Similar to the **(if) Part**.

Lemma 11.

$(l, i, o, x) \in Pred(l', G \wedge [x'])$ if and only if $((l, [x]), i, o) \in aPred((l', [x']), G')$

Proof. This is a direct consequence of Lemmas 8, 9 and 10.