Test Selection for Data-Flow Reactive Systems based on Observations

Omer Nguena-Timo and Antoine Rollet
University of Bordeaux (LaBRI - CNRS)
Talence, France
{*nguena,rollet*}*@labri.fr*

***Abstract*—Conformance testing amounts to verifying adequacy between the behaviors and the specified behaviors of an implementation. In this paper, we handle model-based conformance testing for data-flow critical systems with time constraints. Specifications are described with a formal model adapted for such systems and called Variable Driven Timed Automata (VDTA). VDTA are inspired by timed automata but they use input/output communication variables, allowing clear and short specifications. We present a conformance relation for this model and we propose a symbolic test selection algorithm based on a test purpose. The selection algorithm computes the variations on inputs allowing to reach an expected state of the implementation. Then we propose an on-line testing algorithm.**

***Keywords*-Data-flow systems, Modelling, Timed constraints, Model-based testing, Conformance relation, Symbolic Test Selection.**

I. Introduction

Testing is a popular technique used to increase the quality of systems. 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, etc. Testing techniques may be very different depending on the kind of systems we intend to validate (e.g. embedded systems, concurrent, sequential, reactive, interactive, real-time, communication protocols, etc.). In this work, we consider formal conformance testing for data-flow reactive systems i.e. checking if the behavior of an implementation conforms to its specification described in a formal model.

Data-flow reactive systems continuously compute outputs and internal states according to the inputs and timing constraints. In this framework, continuous means that either the values of the input events can be updated at anytime or the time elapses when no output update is performed. An output update is performed as soon as a constraint is satisfied. So, output updates are eager and performed prior to time elapsing and input updates. The environment can observe outputs when systems are stabilized. Data-flow reactive systems are widely used in the industrial automation domain. Examples of such systems are digital/sequential circuits and control command systems. Testing data-flow reactive systems remains an important challenge.

The success of a formal testing framework depends on specifications. Test cases are selected from the specification and it is better to work with models that allow concise, clear and short specifications. Thus, choosing adequate formalisms for specifying data-flow reactive systems is not a neglected task. To the best of our knowledge, the widespread existing transition systems formalisms are not adequate. Modelling data-flow systems with the *event-based* formalisms (FSM, LTS, EFSM, UPPAAL, models with urgency [1], [2], etc.) often results huge and rather unclear models and *variable-based* formalisms ([3], [4], [5], [6], [5], [7]) do not usually permit to handle dense time.

Contributions. We propose a conformance testing framework for data-flow reactive systems composed of : (1) the formal model called *Variable Driven Timed Automata* (VDTA), (2) a timed variable conformance testing relation (**tvco**), (3) and a test selection algorithm based on test purposes and deterministic specifications. VDTA is a new variable based formalism [8] adapted for specifying data-flow systems with dense time. The inputs and the outputs of systems are variable updates: the environment can assign new values to the input variables and observes the output ones. Transitions are urgent and fired as soon as constraints are satisfied. Outputs occur on transition firing. Input updates or time elapsing occur when no output update is performed. Roughly, an implementation conforms to its specification whenever the observed values of the output variables of the implementation are the same as the ones of the specification after any stabilized sequence of input updates or delays. Test purposes permit to describe behaviors we intend to test. The *test selection algorithm* based on test purposes uses a principle inspired by [9], [10]: (a) we define the *observation product* of the specification and the test purpose; from the observation product, (b) we check existence of a sequence of variable observations and finally (c) we derive test cases. Step (b) uses a reachability analysis.

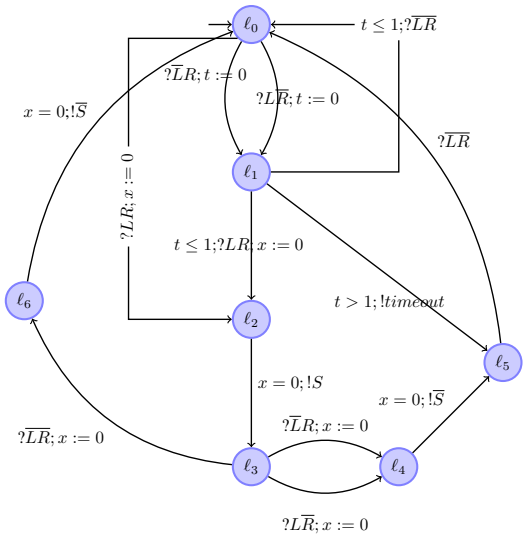
Related work. *Event-based* transition systems formalisms (FSM, LTS, EFSM, Timed Automata, UPPAAL, TEFSM, TIOA, etc.) are widely used for testing synchronous real-time systems [11], [12], [13], [14], [15]. These real-time testing approaches are adaptations of untimed testing approaches based on LTS/ioco theory ([9], [10]). The

event-based formalisms are not adapted for data-flow and variable based communication systems. Events are not persistent (they disappear as soon as they occur) and a single occurrence of an event does not allow to fire several transitions. Each event represents a combination of possible values of flows. From each state and each expected event, it is necessary to add a direct transition to a target state. This phenomenon is broken in the *variable-based* transitions systems: transitions are fired when conditions on variables become true. The Dijkstra Guarded Command Language [3] is a variable-based model adapted for data-flow systems as well as industrial models [4], [5], [6], [5], [7]. But, they do not permit to handle dense time. The IF representation [16] allows to express in a convenient way most useful concepts needed for the specification of asynchronous systems. The representation considers the dense time domain and combines synchronous (with gates) and asynchronous communication (with buffers). VDTA, that is inspired by urgent timed automata [2], can be seen as timed extension of the Dijkstra language [3]. VDTA are not far from a fragment of the IF representation: communication in VDTA uses variables only. In IF, the definition of stable states is based on syntax [16] whereas it is based on semantic in VDTA. This seems more realistic. A sound and exhaustive on-the-fly testing algorithm with VDTA is given in [8]. But this algorithm does not allow to select observable behaviors we want to test. A test purpose based selection algorithm is proposed in [17]; but test purposes are used to describe non observable behaviors.

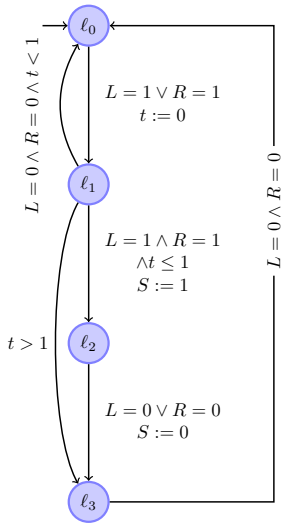
Organization of the paper. The structure of the document is as follows : Section II presents some definitions and notations concerning the VDTA model and its semantics. In Section III, we summarize the principle and the expected results of the reachability analysis of VDTA. Section IV presents the conformance relation called *tvco* and the execution algorithm for a symbolic test case. Section V presents the test selection algorithm based on the definition of the observation product and the reachability analysis. Section VI gives further directions to this work.

II. Variable Driven Timed Automata

Variable Driven Timed Automata (VDTA) are introduced as they allow concise, short and clear specification of data-flow reactive systems. VDTA use the variable-based communication mechanism which allows to restrict the number of transitions in the model. Transitions in VDTA are taken when constraints become true; this allows to fire instantaneously (within zero time unit) more than one transition after a variable value has changed. This is not the case with event-based model where transitions are taken on the occurrence of events. When modelling with the event-based formalism one needs to specify, for each state and each expected event, the target state. We illustrate these



(a) TIOA model for the two buttons machine.



(b) VDTA for the two buttons machine.

Figure 1. Event-based Vs variable-Based Model

statements on models specifying the control program of a "two buttons machine" [18]: *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 less than 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).*

The VDTA in Figure 1(b) is clearer and shorter than the Timed Input Output Automata (TIOA) in Figure 1(a). The TIOA has 4 input events (LR, $\overline{L}R$, $L\overline{R}$, $\overline{LR}$ where L/$\overline{L}$ mean that the left button is pushed/released), 2 output

events (S, $\overline{S}$) and 2 clocks (x, t) whereas the VDTA model has 2 boolean input variables(L and R), 1 boolean output variable (S), and a unique real-valued clock variable (t). The TIOA assumed that in every state the program may receive (denoted by the symbol "?") an event that corresponds a combination of values of L and R before leaving the state. More generally, if there are n buttons (variables) all in domains of size m, one may need to consider up to n^m outgoing edges from each node of the model. This explosion can be reduced using a variable-based modelling approach. The main idea is to hide the synchronisation (between the environment and the system) that happens on variable updates and to concentrate on the functional behaviour of the system that depends on constraints. In the VDTA model in Figure 1(b) one can move instantaneously (through ℓ_1) from ℓ_0 to ℓ_2 if L and R are pressed simultaneously. The description of this behavior in Figure 1(a) has required a direct transition from ℓ_0 to ℓ_2 labelled with LR.

The rest of the section presents formal definitions for VDTA.

A. VDTA Model

Let $\mathbb{N}$, $\mathbb{Q}_+$ and $\mathbb{R}_+$ denote the sets of natural, non-negative rationals and real numbers, respectively. Let $V = \{V_1, \cdots, V_n\}$ 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 $Dom(V) = \Pi_{i\in[1..n]} Dom(V_i)$, 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 $\Pi_{i\in[1..n]}(\{V_i\} \times (Dom(V_i) \cup \{\perp\}))$ and we denote by $A(V)$ the set of variable assignments for V. Given a valuation $v = (v_1, \cdots, v_n)$ 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 $V_i \in V$, $c \in Dom(V_i)$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Given $G \in \mathcal{G}(V)$ and a valuation $v \in Dom(V)$, we write $v \models G$ when $G(v) \equiv true$. As usual, we denote $Proj_{V_i}(G) \in \mathcal{G}(V_i)$ the projection of G over V_i.

Definition 1 *A* Variable Driven Timed Automaton *(VDTA) is a tuple $\mathcal{A} = \langle L, X, I, O, \ell^0, G^0, \Delta_\mathcal{A}\rangle$, where L is a finite set of* locations*, X is a finite set of clocks, I and O are disjoint finite sets of input and output variables, $\ell^0 \in L$ is the* initial location*, $G^0 \in \mathcal{G}(I, O)$ is the* initial condition *with only one solution, a constraint with variables in $I \cup O$ and $\Delta_\mathcal{A} \subseteq L \times \mathcal{G}(I, O, X) \times A(O) \times 2^X \times L$ is the* transition relation

In a transition $\langle \ell, G, A, \mathcal{X}, \ell'\rangle \in \Delta_\mathcal{A}$ (often written $\ell \xrightarrow{G,A,\mathcal{X}} \ell'$): $G \in \mathcal{G}(I, O, X)$ is a boolean combination of elements of $\mathcal{G}(I)$, $\mathcal{G}(O)$ and $\mathcal{G}(X)$; $A \in A(O)$ is an assignment on output variables and $\mathcal{X} \in 2^X$ is a set of clocks that are reset when triggering the transition.

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.

B. Semantics and Notations

A *state* of a VDTA is of the form (ℓ, i, o, x) where $\ell \in L$ is a location, i, o and x are *valuations* of input, output and clock variables. A valuation is simply a function that returns the values of the variables. If $A \in A(I)$ is an assignment on input variables, the valuation $i[A]$ changes the value of input variables according to this assignment. If x is clock valuation, $\mathcal{X}$ is a subset of clocks, and $\delta \in \mathbb{R}_+$ a delay, the valuation $x + \delta$ adds δ to each clock value and the valuation $x[\mathcal{X} \leftarrow 0]$ resets from x all clocks in $\mathcal{X}$.

Definition 2 *The* semantics *of a VDTA $\mathcal{A}$ is a timed transition system $[\![\mathcal{A}]\!] = \langle S_\mathcal{A}, s^0, \Sigma, \rightarrow\rangle$ where $S_\mathcal{A} = L \times Dom(I) \times Dom(O) \times \mathbb{R}_+^X$ is the (infinite) set of* states*, $s^0 = (\ell^0, i^0, o^0, x^0)$ is the initial state 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** $(\ell, i, o, x) \xrightarrow{A} (\ell', i, o[A], x[\mathcal{X} \leftarrow 0])$ *if there exists $(\ell, G, A, \mathcal{X}, \ell') \in \Delta_\mathcal{A}$ such that $(i, o, x) \models G$,*
- **T2** $(\ell, i, o, x) \xrightarrow{A} (\ell, i[A], o, x)$ *with $A \in A(I)$ if $\forall (\ell, G, A', \mathcal{X}, \ell') \in \Delta_\mathcal{A}$, $(i, o, x) \not\models G$.*
- **T3** $(\ell, i, o, x) \xrightarrow{\delta} (\ell, i, o, x + \delta)$ *with $\delta > 0$ if for every $\delta' < \delta$, for every symbolic transition $(\ell, G, \mathcal{X}', \ell') \in \Delta_\mathcal{A}$, we have $(i, o, x + \delta') \not\models G$.*

The semantics considers *discrete transitions (T1 and T2)* and *delay transitions (T3)*. Discrete transitions concern the updates of either input (T2) or output (T1) variables. Delay transitions represent the elapse of time. Output-update transitions are obliged to be fired as soon as constraints are satisfied. Input-update transitions allow to change the

input values only; they are fired by the environment. Input updates and delays occur only when no guard is satisfied (or equivalently no output-update is possible).

Notations. Given a state $s = (l, i, o, x) \in S$, $Out(s) = o$ gives access to the outputs value of $[\![\mathcal{A}]\!]$ in state s. We write $s \xrightarrow{a}$ when there exists s' such that $s \xrightarrow{a} s'$; otherwise we write $s \not\xrightarrow{a}$ For a sequence $\sigma = a_1.a_2.\ldots.a_{k-1}.a_k$ of Σ^*, $s \xrightarrow{\sigma} s'$ if there exists $\{s_i\}_{i=1..k-1}$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots. \xrightarrow{a_{k-1}} s_{k-1} \xrightarrow{a_k} s'$ and we write $s \xrightarrow{\sigma}$ if there exists s' such that $s \xrightarrow{\sigma} s'$. Given a state s of $[\![\mathcal{A}]\!]$, a run is a sequence of alternating states and actions $r = s_0 a_1 s_1 \cdots a_n s_n$ in $S.(\Sigma.S)^*$ such that $\forall i \geq 0, s_i \xrightarrow{a_{i+1}} s_{i+1}$. $Run(s, [\![\mathcal{A}]\!])$ denotes the set of runs that can be executed in $[\![\mathcal{A}]\!]$ starting in state s and we let $Run([\![\mathcal{A}]\!]) = Run(s^0, [\![\mathcal{A}]\!])$. The *trace* $\rho(r)$ of a run $r = s_0 a_1 s_1 \cdots a_n s_n$ is given by the sequence $\rho(r) = Proj_\Sigma(r) = a_1 \cdots a_n \in \Sigma^*$. $Tr([\![\mathcal{A}]\!]) = \{\rho(r) | r \in Run([\![\mathcal{A}]\!])\}$ is the set of traces generated by $\mathcal{A}$. Here are two possible runs of the VDTA in Figure 1(b): $(\ell_0, (0,0), 0, 0) \xrightarrow{L:=1} (\ell_0, (1,0), 0, 0) \xrightarrow{Id_O} (\ell_1, (1,0), 0, 0) \xrightarrow{0.3} (l_1, (1,0), 0, 0.3)$ and $(\ell_0, (0,0), 0, 0) \xrightarrow{L:=1, R:=1} (\ell_0, (1,1), 0, 0) \xrightarrow{Id_O} (\ell_1, (1,1), 0, 0) \xrightarrow{S:=1} (\ell_2, (1,1), 1, 0)$.

C. Deterministic VDTA

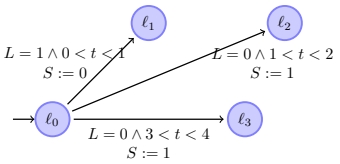
It can happen that the two constraints on two transitions leaving a location are satisfied by the same values of clocks and variables. In this case, the system behaves non-deterministically. A VDTA $\mathcal{A}$ is *deterministic* if G_0 is satisfied by at most one valuation (i^0, o^0); and whenever there exists $\ell \xrightarrow{G_1, A_1, \mathcal{X}_1} \ell_1$, $\ell \xrightarrow{G_2, A_2, \mathcal{X}_2} \ell_2$ with $\ell_1 \neq \ell_2$, we have that $G_1 \cap G_2$ is unsatisfiable.

D. Stable VDTA.

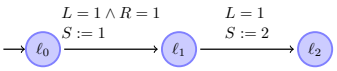
Thanks to their priority, several output-update transitions can be triggered in null delay. A *stable state* is a state from which no output-update transition can be fired. Input-updates and time elapsing are not allowed in non stable states. A state s of $[\![\mathcal{A}]\!]$ is *stable* whenever for every $A \in A(O)$, $s \not\xrightarrow{A}$. To leave this state, input updates or delays are required. Considering Figure 1(b), the states $(\ell_0, (1,0), 0, 0)$ and $(\ell_0, (1,1), 0, 0)$ are not stable; but the state $(\ell_2, (1,1), 1, 0)$ is stable. A *stable run* is a run that ends in a stable state. A VDTA $\mathcal{A}$ is *stable* if there is no loop of unstable states in $[\![\mathcal{A}]\!]$. In the sequel, we shall only consider stable VDTA.

III. Reachability Analysis of VDTA

The reachability analysis consists in checking whether from a source state we can reach a target state. It is required in validation (model-checking) algorithms. In the following, we use the reachability analysis of VDTA in order to generate test cases. Our algorithm will also construct sequences of constraints on input updates and delays that allow to reach the target state. Such sequences can be used as test cases. It is not the purpose of the paper to detail the reachability algorithm for VDTA (it can be found in [17]), but we summarize the results.



(a) A Deterministic VDTA



(b) A VDTA without clocks

Figure 2. Examples of VDTAs

The backward reachability analysis works, in a standard way, as follows: it starts in the set of target states and it computes in each step the predecessors of already encountered states. The algorithm stops when no new state is computed. Let $P \subseteq S_\mathcal{A}$ be a subset of states of $\mathcal{A}$. The computation of the set of predecessors of P (denoted $Pre(P)$) involves the computation of its *output-update predecessors* ($Pre_o(P)$), its *input-update predecessors* ($Pre_i(P)$) and its *time predecessors* ($Pre_t(P)$). The output-update predecessors of P, $Pre_o(P)$ is the set of states from which a state in P can be reached just after an output-update is performed. A similar definition holds for the input-update predecessors. The time predecessors of P, $Pre_t(P)$ is the set of states from which a state in P can be reached by letting the time elapse. Finally the predecessors of P, $Pre(P) = Pre_o(P) \cup Pre_i(P) \cup Pre_t(P)$.

For example, consider the VDTA in Figure 2(a) where L is the only boolean input variable, S is the only boolean output variable and t is the only clock variable. The state $(\ell_0, 0, 0, 2.2)$ is a time predecessor of $(\ell_0, 0, 0, 2.8)$; but $(\ell_0, 0, 0, 1.999)$ is not time predecessor of $(\ell_0, 0, 0, 2.8)$ since $(\ell_0, 0, 0, 1.999)$ is not stable and the transition to ℓ_2 is taken prior to time elapsing. $(\ell_0, 0, 0, 1.999)$ is an output-update predecessor of $(\ell_2, 0, 1, 1.999)$.

Since the number of states is infinite, the algorithm may not terminate. So, we compute predecessors on symbolic states. Since the number of encountered symbolic states is finite, the algorithm terminates. A symbolic state is of the form (ℓ, Z_I, Z_O, Z_X) where Z_I, Z_O and Z_X are constraints over input, output and clock variables. Z_X may represent the (un)famous clock region [19] like in [17] or clock zone [20]. Following the semantics, a symbolic state has a set of input-update predecessor, output-update predecessors, and time-elapsing predecessors. The paper does not focus on the computation of the predecessors which is effective and rather technical. We are interested in the resulting reachability graph.

Reachability Graph. The reachability graph is a kind of

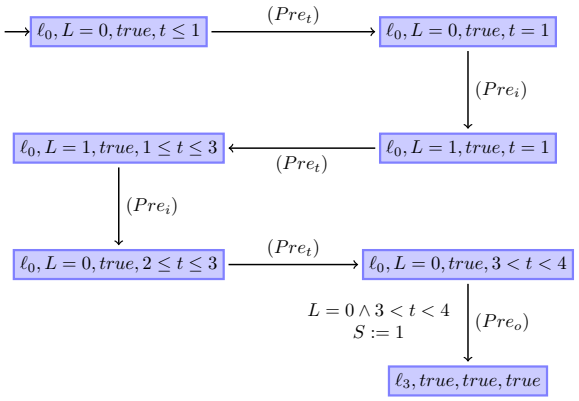


Figure 3. A path in a reachability graph.

VDTA where locations are tagged with invariants. Locations are symbolic states and invariants are constraints on variables and clocks. We construct a *reachability graph* during the reachability analysis. Let us consider, $Z[A] = \{x[A] \mid x \in Z\}$ is set of valuations from which we can reach a valuation satisfying Z after an update A is executed and $Z\downarrow = \{x \mid \exists\ \delta \in \mathbb{R}_+,\ \text{s.t}\ x + \delta \in Z\}$ is the set of valuations (the zone) from which clock valuations in Z can be reached by letting the time elapse. The construction works as follows: if $(\ell', Z'_I, Z'_O, Z'_X) \in Pre(\ell, Z_I, Z_O, Z_X)$ then, the transition $(\ell', Z'_I, Z'_O, Z'_X) \xrightarrow{G,A,\mathcal{X}} (\ell, Z_I, Z_O, Z_X)$ exists if there are $\ell' \xrightarrow{G,A,\mathcal{X}} \ell \in \Delta_{\mathcal{A}}$, $A' \in A(I)$ such that $Z'_I[A] \subseteq Z_I$, $Z'_O \subseteq Z_O$ and $Z'_X \cap (Z_X\downarrow) \neq \emptyset$; otherwise there is no label on the transition from $(\ell', Z'_I, Z'_O, Z'_X)$ to (ℓ, Z_I, Z_O, Z_X).

Let us give the intuitive semantics of the reachability graph: one can stay in a location only if the invariant holds. The invariant can be violated provided that a transition is fired after the violation. The invariant must be satisfied when entering a location. A transition in the graph asserts that we can move from the source to the target location provided that the constraint is satisfied.

Figure 3 presents a path (the whole graph is too large) in the reachability graph we can construct from the model in Figure 2(a) using the clock zone abstraction. We assumed that $(\ell_3, true, true, true)$ and $(\ell_0, L = 0, S = 0, t = 0)$ are the target and source locations. $s_1 = (\ell_0, L = 0, true, t = 1)$ is the input predecessor of $s_2 = (\ell_0, L = 1, true, t = 1)$. Note that $s_2 \in Pre_i(s_1)$ but this is not represented in the path in Figure 2(a).

IV. Conformance Testing With VDTA

Conformance testing consists in checking that an implementation exhibits an observable behavior consistent with its specification. The main idea of our conformance relation is that all observable behaviors of the implementation have to be allowed by the specification. Especially:

1) An implementation is not allowed to update a variable in a time (too late or too early) when it is not specified.
2) An implementation is not allowed to omit to change the values of variables at the time it is specified.

A. Observations and Conformance Testing Relation

In the context of VDTA, it is not obvious when output observations should be performed. Transitions are urgent and several transitions can be fired in null time. There are two directions for observing the outputs: either one observes the outputs after each transition firing as it is done for *tioco* [14], or one observes the outputs when the implementation has reached a *stable* state. We consider that the environment and the implementation run with the same speed implying that the tester can not observe a value of a variable if it lasts zero time unit. In this case we observe the outputs in stable states only.

Given a stable state s, we consider the next stable states (or a singleton in case of determinism) the implementation can reach from s after the execution of an input-update $A_i \in A(I)$. Given two stable states $s, s' \in S$, we write:

- $s \overset{A_i}{\Longrightarrow} s'$ if there exists a sequence $\sigma \in A(O)^*$ s.t $s \xrightarrow{A_i} s'' \xrightarrow{\sigma} s'$, i.e. s' is a stable state that can be reached from s after updating the input variables with A_i, only triggering urgent transitions in zero time unit.
- $s \overset{\delta}{\Longrightarrow} s'$ if there is a sequence $\sigma = \sigma_1 \cdots \sigma_n \in (\{Id_O\} \cup \mathbb{R}_+)^*$ s.t $s \xrightarrow{\sigma} s'$ and $\delta = \sum_{\delta_i \in Proj_{\mathbb{R}}(\sigma)} \delta_i$, i.e. s' is a stable state that can be reached by letting the time elapse during δ units of time with no observable output update.

The timed transition system $Obs(\mathcal{A}) = (S, s^0, A(I) \cup \mathbb{R}_+, \Longrightarrow)$ is inductively generated from $[\![\mathcal{A}]\!]$ by starting from s^0 (that is supposed to be stable) and by using the two previous rules. We let $ObsRun(\mathcal{A}) = Run(Obs(\mathcal{A}))$ and $ObsTr(\mathcal{A}) = Tr(Obs(\mathcal{A}))$ denote the set of observed runs and observed traces of $\mathcal{A}$. Note that a trace in $ObsTr(\mathcal{A})$ is a sequence of input updates and delays. Finally, we define s *Safter* $\alpha = \{s' \mid s \overset{\alpha}{\Longrightarrow} s'\}$ and $\mathcal{A}$ *Safter* $\alpha = s^0$ *Safter* α. For example, using Figure 1(b): $(\ell_0, (0,0), 0, 0)$*Safter* $\{0.3.(L := 1; R := 1)\} = \{(\ell_2, (1,1), 1, 0)\}$.

Now we define the conformance testing relation **tvco** (i.e timed variable conformance relation) adapted for VDTA. We assume that the implementation Imp and the specification $\mathcal{A}$ are both modeled by compatible VDTA (i.e. they share the same input and output variables).

Definition 3

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

where $Out(.)$ *returns the values of the output variables.*

Intuitively, the relation intends to check if the values of output variables of the implementation are correct after any stabilized sequence made of input assignments or time elapsing. The tester can observe all output variables of the implementation and can only update the input variables of the implementation or let the time elapse. Remark that the blocking aspect as in [9] is not considered so far.

Comparison with *tioco*. In [14], the **tioco** relation allows to observe all the outputs of consecutive transitions, even those that are fired in null delay. The relation **tvco** only considers the outputs that are observable in the next stabilized states, which seems more realistic for us.

B. Testing With a Symbolic Test Graph

A *symbolic test graph* is like a deterministic VDTA except that its locations are tagged with invariants. Invariants are constraints on variables and clocks. A symbolic test graph has two special locations denoted $Pass$ and Inc (for inconclusive). For example, Figure 3 is a symbolic test graph assuming that the trap location ℓ_3 is a $Pass$ location.

The conformance testing algorithm of an implementation Imp with respect to a test graph works as follows: the algorithm starts at the initial state. The tester performs either an input update or delay for a chosen duration δ. An input-update or a delay is performed provided that they do not violate the invariant of the current location except if they allow to fire a transition of the test graph. After an input-update is performed, the tester compares the outputs of the implementation with the outputs of the current state of the test graph. It returns the verdict "fail" when they do not conform. While delaying for δ time units, the tester observes the outputs and checks their conformance. If the location Inc is reached, it means that a non relevant behavior is being tested. If the location $Pass$ is reached, it means that a specified behavior has been successfully tested; the tester returns the verdict "pass" and it tries to test another behavior. This algorithm is inspired by the on-line testing algorithm in [8]. For example, the graph in Figure 3 can be used for testing the reachability of ℓ_3 in the VDTA of figure 2(a) and the following test case (scenario) can be generated: let the time elapse until $t = 1$, and set L to 1; then let the time elapse and set L to 0 when t is between 3 and 4.

We consider in a usual way that an implementation Imp **passes** a test case if and only if any test run leads to a $pass$ verdict.

V. Test Selection with Test Purpose

The test selection algorithm is based on the notion of *Test Purpose* (TP) and it follows the methodology described in [9], [10]. Given a specification $\mathcal{A}$ and a test purpose TP, we construct a test graph from which we select symbolic test cases. A test graph results from the *observation product* (it differs from the synchronous product due to the stabilization aspects) of the specification with the test purpose. The selection algorithm analyses the reachability of a location $Pass$ in the test graph.

A. Test Purpose

In practice, a test purpose allows to select some *observable* behaviors of the specification we want to test. A test purpose is modeled by a VDTA. We consider that a test purpose is *non intrusive* with respect to the specification: it is not allowed neither to reset clocks of the specification nor to assign new values to the output variables. But a test purpose is equipped with its own set of clocks that allow to time-stamp and to check the observations.

Definition 4 *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* $TP = \langle S, X \cup X', I, O, s^0, G^0, \Delta_{TP} \rangle$ *such that:* S *is a finite set of locations with a mandatory special trap location* $Accept_{TP}$ *and an* optional *special trap location* $Reject_{TP}$*;* s^0 *is the initial location;* I*,* O *and* X *are respectively the input, output and clock variables of the specification;* $G^0 \in \mathcal{G}(I, O)$ *is the* initial condition *of* TP *and* $\mathcal{A}$*;* X' *is the set of* private clocks *of* TP*, with* $X' \cap X = \emptyset$*; and* $\Delta_{TP} \subseteq S \times \mathcal{G}(I, O, X, X') \times Id_O \times 2^{X'} \times S$ *is the transition relation.*

As for the specification, we assume that the guards in $\mathcal{G}(I, O, X, X')$ are given by a boolean combination of elements of $\mathcal{G}(I)$, $\mathcal{G}(O)$, $\mathcal{G}(X)$ and $\mathcal{G}(X')$. Note that TP is allowed to observe the states (the values of variables) of $\mathcal{A}$.

Intuitively, a location of a test purpose allows to observe the values of clocks and variables (in stable states of the implementation) and checks whether they satisfy constraints on transitions leaving the location. When a constraint is satisfied, the corresponding transition is passed and the location changes. As in [9], [10], a location of a test purpose acts like the *Until* operator of the temporal logic LTL (stay in the location until a constraint becomes true). A path from s^0 to $Accept_{TP}$ specifies the observations we want to test while a path to $Reject_{TP}$ specifies non relevant observations that we do not want to test. According to the semantics of VDTAs, 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.

Figure 4 presents three examples of test purposes. Test purposes in Figure 4(a) and 4(b) observe variables of the VDTA and they only specify which behaviors of the implementation are interesting for the test. The test purpose in Figure 4(c) has its own clock variable x; it specifies which behavior of the implementation should be tested, but also the behavior of the implementation that should not be tested (from the location $Reject_{TP}$, the location $Accept_{TP}$, cannot be reached anymore). Let us comment the three test

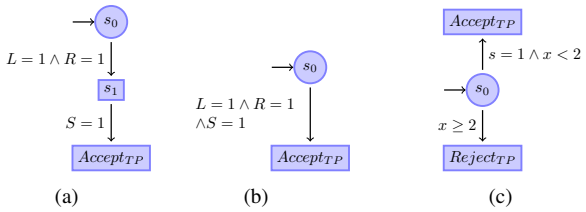


Figure 4. Test purposes.

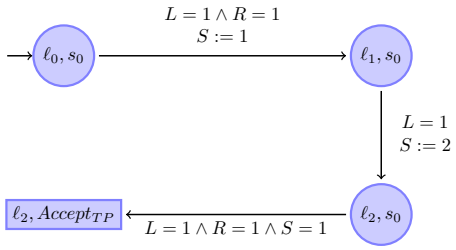


Figure 5. The observation product of Figure. 2(b) Figure. 4(b)

purposes:

- The one in Figure 4(a) requires that if L and R equal 1 in the same time during the test process, then the variable S equals to 1 in the future.
- The one in Figure 4(b) requires to have s equal to 1 in the same time that L and R equal 1.
- The one in Figure 4(c) requires to test the behaviors of the implementation in which s is set to 1 at most 2 time units after the beginning of the session.

B. Observation Product

Since we use a test purpose to select the observable behaviors we want to test, we consider an operation that allows to observe and to check the values of clocks and variables when the specification (implementation) is in stable states only. We call this operation the *observation product*.

Given a location ℓ of $\mathcal{A}$, $G_{\mathcal{A}}(\ell)$ denotes the set of constraints on outgoing transitions of ℓ and it is defined by $G_{\mathcal{A}}(\ell) = \{G \in \mathcal{G}(I,O,X) \mid \exists \ell' \in L,\ \ell \xrightarrow{G,A,\mathcal{X}} \ell'\}$.

Definition 5 *Let $\mathcal{A} = \langle L, X, I, O, \ell^0, G^0, \Delta_{\mathcal{A}}\rangle$ be a specification and let $TP = \langle S, X' \cup X, I, O, s^0, G^0, \Delta_{TP}\rangle$ be a test purpose. The* observation product *of $\mathcal{A}$ and TP is the VDTA $\mathcal{A} \otimes TP = \langle L \times S, X \cup X', I, O, (\ell^0, s^0), G^0, \Delta_{\mathcal{A}\otimes TP}\rangle$ where the rules $\mathcal{R}_1$ and $\mathcal{R}_2$ define $\Delta_{\mathcal{A}\otimes TP}$:*

$\mathcal{R}_1$: $(\ell, s) \xrightarrow{G_\ell, A, \mathcal{X}} (\ell', s)$ *iff there is* $\ell \xrightarrow{G_\ell, A, \mathcal{X}} \ell'$

$\mathcal{R}_2$: $(\ell, s) \xrightarrow{G \wedge G_s, Id_O, \mathcal{X}'} (\ell, s')$ *with* $G = \bigwedge_{G' \in G_{\mathcal{A}}(\ell)} \neg G'$ *iff there is* $s \xrightarrow{G_s, Id_O, \mathcal{X}'} s'$

We shall denote $Accept$ the set of states of $\mathcal{A} \otimes TP$ of the form $(\ell, Accept_{TP})$.

Intuitively, in a location (ℓ, s) (or a state $((\ell, s), i, o, x)$) of $\mathcal{A} \otimes TP$ we want to observe and check the values of clocks and variables. There are two situations: the first situation, described by $\mathcal{R}_1$, occurs when we can not observe the values because the specification is in a non-stable state. In this case, only the system is allowed to fire an urgent transition. In the second situation, described by $\mathcal{R}_2$, the state of the specification is stable (no constraint on transitions leaving the location is satisfied) and we observe the values. Then, we check the observations and a transition of the test purpose can be passed.

For example, Figure 5 presents the observation product of the specification in Figure 2(b) with the test purpose in Figure 4(b). The location (ℓ_0, s_0) has one outgoing transition since when L and R equal 1 the transition from ℓ_0 is urgently taken, no matter the value of S. Thus, the corresponding state of the specification (or the system) is non-stable and consequently variables can not be observed. Since variables can not be observed, we can not check whether the constraint $L = 1 \wedge R = 1 \wedge S = 1$ on the transition leaving s_0 holds. The same principle applies to every location of the observation product. Note that conformance is tested only when TP has reached its stable states.

In [17], TP was used to describe non observable behaviors. The notion of *synchronous product* in [17] is no longer adequate for this framework since TP only describes observable behaviors.

C. Test Graph Construction

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

Step 1. We perform the observation product $\mathcal{A}_{TP} = \mathcal{A} \otimes TP$ in order to characterize in $\mathcal{A}$ the behaviors the test purpose requires to test.

Step 2. The reachability analysis applied to $\mathcal{A}_{TP}$, returns a reachability graph. The test graph is obtained from the reachability graph by transforming every $Accept$ location to $Pass$ and every $Reject$ location to Inc.

Discussion. The test graph constructed with the above algorithm has no *Fail* location. This is a difference with the construction in [9], [10]. The reason is that the backward reachability analysis algorithm does not determine the exact value of the outputs in locations of the test graph. In addition to the backward analysis that allows to compute location invariant, a forward reachability analysis can be performed to have outputs values in locations. However, performing this analysis before running the testing algorithm can be expensive. Our testing algorithm presented in Subsection IV-B performs the forward analysis during the test and it computes the *Fail* locations on-the-fly.

VI. Concluding Remarks

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. We have proposed the new timed variable conformance testing relation (**tvco**) adapted to the model, and a test generation method with a selection based on a test purpose.

As a future work, we intend to consider blocking aspect in systems. Note that we are currently working on a real industrial case study: a "flashmanager" application. First results are promising. Besides, we also intend to consider assignments of variables with delicate operations (e.g. $x := y + 3$). Such tough operations will increase the expressive power of VDTA.

Acknowledgment

This work has been supported by the French ANR Testec Project. We also would like to thank the INRIA Vertecs team for their comments.

References

[1] S. Bornot, J. Sifakis, and S. Tripakis, "Modeling urgency in timed systems," in *Proc. of COMPOS'97*, ser. LNCS. Springer, 1998, pp. 103–129.

[2] R. Barbuti and L. Tesei, "Timed automata with urgent transitions," *Acta Inf.*, vol. 40, no. 5, pp. 317–347, 2004.

[3] E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Commun. ACM*, vol. 18, no. 8, pp. 453–457, 1975.

[4] L. D. Bousquet, F. Ouabdesselam, J.-L. Richier, and N. Zuanon, "Lutess: A specification-driven testing environment for synchronous software," in *Proc. ICSE'99*, 1999, pp. 267 –276.

[5] B. Marre and A. Arnould, "Test sequences generation from lustre descriptions: Gatel," in *Proc. of ASE'00*. IEEE, 2000, p. 229.

[6] P. Raymond, X. Nicollin, N. Halbwachs, and D. Waber, "Automatic testing of reactive systems," in *Proc. of RTSS'98*. IEEE, 1998, pp. 200–209.

[7] B. Seljimi and I. Parissis, "Using clp to automatically generate test sequences for synchronous programs with numeric inputs and outputs," in *Proc. of ISSRE'06*. IEEE, 2006, pp. 105–116.

[8] O. Nguena-Timo and A. Rollet, "Conformance testing of variable driven automata," in *Proc. of WFCS'10*. IEEE, 2010, pp. 241–248.

[9] J. Tretmans, "Test generation with inputs, outputs, and repetitive quiescence," *Software-Concepts and Tools*, vol. 17, pp. 103–120, 1996.

[10] C. Jard and T. Jéron, "Tgv: theory, principles and algorithms," *Int. J. Softw. Tools Technol. Transf.*, vol. 7, no. 4, pp. 297–315, 2005.

[11] R. Cardell-Oliver, "Conformance testing of real-time systems with timed automata specifications," *Formal Aspects of Computing Journal*, vol. 12, no. 5, pp. 350–371, 2000.

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

[13] M. Krichen and S. Tripakis, "Black-box conformance testing for real-time systems," in *Proc. of SPIN*, ser. LNCS, vol. 2989. Springer, 2004, pp. 109–126.

[14] M. Mikucionis, K. G. Larsen, and B. Nielsen, "T-uppaal: Online model-based testing of real-time systems," in *Proc. of ASE'04*. IEEE, 2004, pp. 396–397.

[15] M. Núñez and I. Rodríguez, "Conformance testing relations for timed systems," in *Proc. of FATES'05*, ser. LNCS, vol. 3997. Springer, 2005, pp. 103–117.

[16] M. Bozga, J.-C. Fernandez, L. Ghirvu, S. Graf, J.-P. Krimm, and L. Mounier, "If: An intermediate representation and validation environment for timed asynchronous systems," in *Proc. of FM '99*. Springer, 1999, pp. 307–327.

[17] O. Nguena-Timo, H. Marchand, and A. Rollet, "Automatic test generation for data-flow reactive systems with time constraints," in *Proc. of ICTSS (Short paper) 2010*. CRIM-Canada, 2010, pp. 25–30.

[18] H. B. Mokadem, B. Bérard, P. Bouyer, and F. Laroussinie, "A new modality for almost everywhere properties in timed automata," in *Proc. of CONCUR'05*, ser. LNCS, vol. 3653, 2005, pp. 110–124.

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

[20] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithm and tools," in *Lectures on Concurrency and Petri Nets*, ser. LNCS, vol. 3098. Springer, 2004, pp. 87–124.