# A Zone-based Reachability Analysis of Variable Driven Timed Automata

Omer Nguena-Timo
University of Bordeaux (LaBRI - CNRS)
Talence, France
nguena@labri.fr

Antoine Rollet
University of Bordeaux (LaBRI - CNRS)
Talence, France
rollet@labri.fr

***Abstract*—In this paper, we propose an algorithm for efficient reachability analysis of Variable Driven Timed Automata (VDTA). VDTA is a new timed behavioural model for data-flow reactive systems in which inputs and outputs are modelled by variables. Such an approach is commonly used in the industry. Reachability analysis is often the basis of model-checking, test generation (especially Test Purpose approach), or control algorithms. For example, a model-based testing framework with VDTA derives test cases by performing a region-based reachability analysis. Thus, an efficient analysis is needed. We propose an algorithm based on the zone abstraction. The algorithm not only checks the reachability, but it also computes timed sequences of input updates required to reach a target. In practice, it is more efficient than region based one.**

***Keywords*-reachability analysis, zone, timed systems, data-flow, urgent edges.**

## I. Introduction

In the context of reliable systems design, formal methods provide a rigorous framework for modelling and reasoning about the systems. Formal reasoning includes methods for testing [15], model-checking [9], supervising [14], etc., the systems, which are commonly and efficiently modelled in a state-transition/automata formalism in which systems behaviours are represented by sequences of (constrained/guarded) transitions between states. Reachability analysis amounts to checking whether a target state is reachable from an initial one. A target state may represent a failure of the system that may cause considerable damages. Reachability analysis is often involved in many validation methods. For example, the generation of test cases with test purposes usually consists in analysing the reachability of certain states and to compute the sequence of actions permitting to reach them. The paper presents a new efficient algorithm for the reachability analysis of Variable Driven Timed Automata [12].

### A. Variable Driven Timed Automata (VDTA)

VDTA [12] is a new timed model adapted for modelling and reasoning about data-flow reactive systems in which input and output are rather modelled by variables. A VDTA model-based testing framework has been developed in [11], [13].

A VDTA is a guarded edge (transition) system in which every edge is labelled with an update of output variables and a constraint on input, output and clock variables. VDTA is inspired by urgent timed input/ouput automata [2], [3]. VDTA implements three main mechanisms. The first mechanism, which is not new, implemented in VDTA is *urgent edges*. As with urgent timed automata [3], edges in VDTA are urgent meaning that they are fired as soon as their constraints become true. Only output variable updates are performed on edges firings. It is well-known [3] that urgency mechanism may allow short and clear specification. Secondly, VDTA implements the *variable based communication mechanism*. This supposes that systems communicate with their environment through input and output variables or sockets. This is closer to how engineers think and how open systems are specified. The VDTA model assumes that the environment freely updates the input variables and only the systems can update the output variables according to their states and timing information given by clock variables. VDTA allows to specify explicitly the events to the environment only. Unlike [2], [3], [8], the events from the environment are not explicitly specified. Thirdly, VDTA is a *variable driven model*. Edges firings in VDTA do not depend on occurences of synchronizing actions but rather on the truth value of constraints only. The values of the variables are persistent and they last until they are updated. Consequently, a single input variable change can trigger instantaneously a chain of consecutive edges in VDTA. This is not the case with event-based formalisms like [3], [8] where each edge firing is provoked by an occurrence of a (non persistent) synchronising action.

The VDTA-based testing method [11], [13] is based on test purposes. Test purposes are VDTA with special states labelled with ACCEPT; they allow to guide the test selection. Roughly speaking, the method consists of two main steps: (1) a test graph is constructed from the specification and a test purpose and then (2) the reachability of the states labelled with ACCEPT is checked. A VDTA reachability analysis algorithm is proposed in [11]. This algorithm is inspired by a seminal reachability algorithm for timed automata [2]. It uses the symbolic region [2] representation of the time, which allows elegant reasoning on timed systems. Since it is a precise discrete representation of clock valuations, it allows to abstract a timed model into a discrete model making thus easier the analysis of urgent edge and unspecified input updates in VDTA. In the backside, region abstraction is expensive. As for the analysis of timed automata [4], [7], we wondered how the symbolic zone abstraction could be used for the reachability analysis of VDTA. Roughly speaking, zones are larger abstractions of clock valuations: one zone can be decomposed into an equivalent finite set of regions. Thus a single computation over a zone may need many computations over the regions it includes. Consequently, the region abstraction is in practice less efficient than the zone abstraction [2], [6]. So, providing an efficient zone-based analysis for VDTA will improve the VDTA test generation algorithm [11], [13].

## B. Contributions

We propose a backward zone-based reachability analysis algorithm for VDTA. The general idea is presented in Algorithm 1. Contrary to the forward one, it starts in the target states $P_0$ and iteratively visits *predecessors* of states until a fixpoint is reached and no new states is computed. $P_0$ is reachable if the fixpoint include the initial state.

**Algorithm 1** Principle of the Backward Analysis

```
$P_0 \leftarrow P$
**repeat**
    $P_{i+1} \leftarrow P_i \cup Predecessor(P_i)$
**until** $P_{i+1}$ equal to $P_i$
```

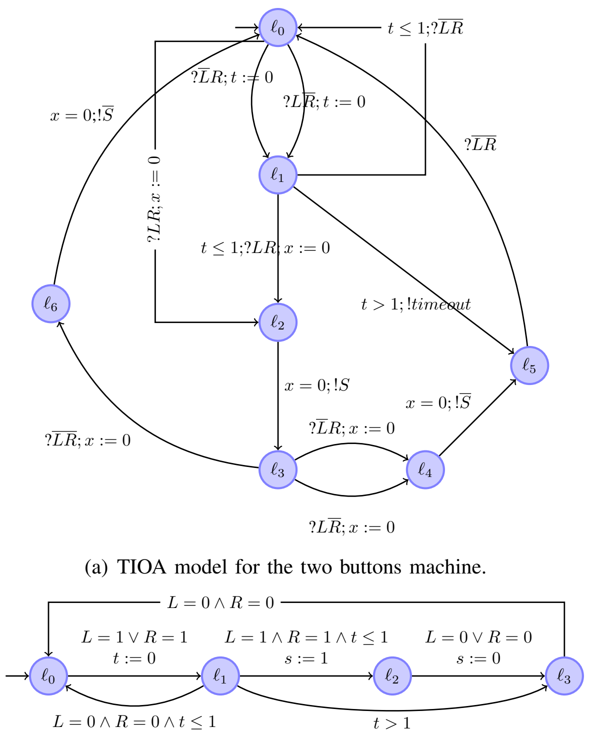
The predecessors of a state of a VDTA include its *input updates predecessors*, *output updates predecessors* and *time predecessors*. To ensure its termination, our algorithm rather works on symbolic states. Input, output and clock values are represented by zones in symbolic states. A similar method is used for timed automata [2], [3], [5], [7]. But the zone-based backward reachability analysis for VDTA could not be a simple adaptation of the zone-based reachability of timed automata. Besides, we found that the analysis of VDTA has many common points with the *analysis of timed game automata (TGA)* [6]. But TGA analysis algorithm cannot work for VDTA. Nevertheless, as for the analysis of TGA we consider a notion of safe time predecessor in order to compute predecessors of symbolic states.

## C. Outline

Section II presents the VDTA model and its semantics. In Section III we define symbolic states and the computation of their predecessors. At the end of the section we present the reachability algorithm and we discuss about its termination. Section IV concludes the paper giving future works with VDTA.

## II. Variable Driven Timed Automata (VDTA)

Before we provide a formal definition for VDTA, we consider the following specification of the control program designed to start a "two buttons machine" [10]:
*The machine starts 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 greater than 1 time unit is performed (time-out has occurred), then the whole process must be started again. After the machine has started*



(a) TIOA model for the two buttons machine.

(b) VDTA for the two buttons machine.

Fig. 1. Event-based Vs variable-Based Model

*(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 to 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 consider up to $n^m$ outgoing edges for each location. 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. Even if there is not an explicit edge from $\ell_0$ to $\ell_2$ in Figure 1(b), one can move instantaneously (through $\ell_1$) from $\ell_0$ to $\ell_2$ if $L$ and $R$ are pressed simultaneously. Such a behaviour has required an explicit edge from $\ell_0$ to $\ell_2$ in Figure 1(a).

However, the VDTA formalism would not have been advantageous in situations where the behaviours depend on former states of the buttons. This may occur when a rising edge of the button needs to be captured and processed even if the button has been released again in the meantime; in such a situation, we would probably need to keep the state "button was pressed" and use a TIOA model.

Anyway, modelling input/output by variables is quite ordinary for engineers. This allows short and clear specification in some circumstances.

## *A. Model and Semantics of VDTA*

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 $\mathcal{G}(V)$ (resp.$\mathcal{G}^+(V)$) the set of variable constraints defined as conjunction (resp. 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 $G(v) \equiv true$ and we define $[\![G]\!] = \{v \in Dom(V) \mid v \models G\}$. For a subset $I$ of $V$, we denote $G_I$ *the projection* of $G$ over the variables in $I$. To ease the notation, if $I$, $O$, are two sets of different sorts of variables, a constraint $G$ in $\mathcal{G}^+(I, O)$ is a boolean combination of a constraint $G_I \in \mathcal{G}^+(I)$, and a constraint $G_O \in \mathcal{G}^+(O)$. This generalises to an arbitrary number of sets of variables.

*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 set of *edges*.
In an edge $\langle \ell, G, A, \mathcal{X}, \ell'\rangle \in \Delta_\mathcal{A}$ (often written $\ell \xrightarrow{G,A,\mathcal{X}} \ell'$): $G \in \mathcal{G}(I, O, X)$; $A \in A(O)$ is an assignment on output variables and $\mathcal{X} \in 2^X$ is a set of clocks that are reset when passing the edge. There is no explicit assignment on input variables.

A *state* of a VDTA $\mathcal{A}$ is of the form $(\ell, i, o, x)$ 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. For example, $(\ell_0, (0, 0), 1, 0.5)$ is a state of the VDTA in Figure 1(b) where $(0, 0)$ is the valuation of the inputs $L$ and $R$, $S$ equals $1$ and $t$ equals $0.5$.

If $A \in A(I)$ is an assignment on input variables, the valuation $i[A]$ changes the value of input variables according to the 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 $\delta$ to each clock value and the valuation $x[\mathcal{X} \leftarrow 0]$ resets from $x$ all clocks in $\mathcal{X}$. For example, if $i = (0, 0)$ and $A = \{L := 1; R := 1\}$ is an assignment over $L$ and $R$ then $i[A]$ is the valuation $(1, 1)$.

*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 $G^0$, $\Sigma = A(I) \cup A(O) \cup \mathbb{R}_+$ 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 continuous transitions (T3). *Output-update transitions* of type T1 allow to update the output variables. Output-update transitions correspond to edges passing. Edges are passed as soon as their constraints are satisfied. *Input-update transitions* of type T2 allow to update the input variables. *Time-elapsing transitions* of type T3 represent the continuous elapse of time.

*Definition 3:* A *stable state* is a state from which no output-update transition can be fired.

Note that input-updates and time-elapsing transitions are allowed stable states only; they are not allowed in non stable states. Input-update transitions allow to change the inputs. They are fired by the environment. Considering Figure 1(b), $(\ell_0, (1, 0), 0, 0)$ is not stable whereas $(\ell_0, (0, 0), 0, 0)$ is stable. Consequently, the only transition from $(\ell_0, (1, 0), 0, 0)$, $(\ell_0, (1, 0), 0, 0) \xrightarrow{Id_O} (\ell_1, (1, 0), 0, 0)$ corresponds to the edge $\ell_0 \xrightarrow{L=1 \vee R=1; t:=0} \ell_1$ $(\ell_0, (0, 0), 0, 0)$ whereas from $(\ell_0, (0, 0), 0, 0)$ one can perform input-update transitions or time-elapsing transitions like: $(\ell_0, (0, 0), 0, 0) \xrightarrow{L=1} (\ell_0, (1, 0), 0, 0)$, $(\ell_0, (0, 0), 0, 0) \xrightarrow{0.3} (\ell_0, (0, 0), 0, 0.3)$ or $(\ell_0, (0, 0), 0, 0) \xrightarrow{0.7} (\ell_0, (0, 0), 0, 0.7)$.

*Definition 4:* A run of $\mathcal{A}$, $r = s_0 a_1 s_1 \cdots a_n s_n$ in $S_\mathcal{A}.(\Sigma.S_\mathcal{A})^*$ is a sequence of alternating states $s_i \in S_\mathcal{A}$ and actions $a_i \in \Sigma$ with $\Sigma = A(I) \cup A(O) \cup \mathbb{R}_+$ such that $\forall i \geq 0, s_i \xrightarrow{a_{i+1}} s_{i+1}$.

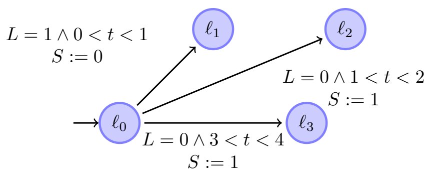


Fig. 2. A VDTA

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 $(\ell_0, (0,0), 0, 0) \xrightarrow{L:=1, R:=1} (\ell_0, (1,1), 0, 0) \xrightarrow{Id_O} (\ell_1, (1,1), 0, 0) \xrightarrow{s:=1} (\ell_2, (1,1), 1, 0)$.

## B. Principle of Reachability Analysis

The *reachability analysis* amounts to checking whether there exist a run that goes through a given target state. We want that the VDTA reachability analysis algorithm returns how and when to modify the inputs in order to reach the target state eventually.

Let us consider the small running example VDTA in Figure 2 where $L$ is the only boolean input variable, $S$ is the only boolean output variable and $t$ is the only clock variable. Now assume that one wants to check the reachability of the location $\ell_3$ whatever are the values of $L$, $S$ and $t$ in $\ell_3$. We want the reachability algorithm to return the verdict "yes" and the following scenario:

1) From the initial state $(\ell_0, 0, 0, 0)$, keep the value of $L$ unchanged and let the time elapse until $t$ equals 1.
2) When $t$ equal 1, set $L$ to 1; then let the time elapse.
3) set $L$ to 0 after the value of $t$ has passed 3 and before it reaches 4.

But if one wants to check the reachability of $\ell_3$ when $S$ equals 0, the algorithm should return that the state is not reachable since $S$ equals 1 after the edge from $\ell_0$ to $\ell_3$ is taken.

Let *Goal* be a set of *states* of $\mathcal{A}$ we want to check the reachability of. Algorithm 1 gives the principle of the backward reachability analysis for $\mathcal{A}$: it starts in *Goal* and computes in each step the predecessors of already encountered states. The algorithm stops when no new state is computed.

*a) Predecessors:* 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; formally, $Pre_o(P) = \{(\ell, i, o, x) \mid \exists\, A \in A(O)\ \exists\, (\ell', i', o', x') \in P \text{ s.t } (\ell, i, o, x) \xrightarrow{A} (\ell', i', o', x')\}$. None of the states in $Pre_o(P)$ is stable.

Similarly, we define the *input-update predecessors* of $P$, $Pre_i(P) = \{(\ell, i, o, x) \mid \exists A \in A(I),\ \exists\ (\ell', i', o', x') \in P \text{ s.t } (\ell, i, o, x) \xrightarrow{A} (\ell', i', o', x')\}$. All the states in $Pre_i(P)$ are stable.

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. Formally, $Pre_t(P) = \{(\ell, i, o, x) \mid \exists \delta > 0,\ \exists\ (\ell', i', o', x') \in P \text{ s.t } (\ell, i, o, x) \xrightarrow{\delta} (\ell', i', o', x')\}$. All the states in $Pre_t(P)$ are stable.

Finally, $Pre(P) = Pre_o(P) \cup Pre_i(P) \cup Pre_t(P)$.

For example, let us consider the VDTA in Figure 2. 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 a time predecessor of $(\ell_0, 0, 0, 2.8)$ since $(\ell_0, 0, 0, 1.999)$ is not stable and the transition to $\ell_2$ is taken prior to time elapsing. $(\ell_0, 0, 0, 1.999)$ is an output-update predecessor of $(\ell_2, 0, 1, 1.999)$.

An execution of Algorithm 1, which computes predecessors of states, may not terminate. This is because VDTA has infinitely many states and each computation step may return a new state. Thus, we consider symbolic representations for states or shortly *symbolic states*; then we describe the computation of predecessor ($Pre_o$, $Pre_i$, $Pre_t$ and $Pre$) of symbolic states. Later, we show that the number of symbolic states encountered during the execution of the algorithm is finite. This will ensure its termination.

*b) Relation with other models:* A similar zone based analysis method is used for timed automata [2], [3], [5], [7]. But it cannot be simply adapted for VDTA. The output predecessor operator over symbolic states of VDTA works like the discrete predecessor operator for timed automata whereas the input and time predecessors do not. In particular, the computation of the time predecessor should not return time instants that enable an urgent edge. Similarly, the input predecessors can not return input data that enable urgent edges. Besides, the analysis of VDTA has many common points with the *analysis of timed game automata (TGA)* [6]. Indeed, since input-update are freely performed by the environment, they are analogous to "controllable actions" in TGA. Output update transitions are analogous to "uncontrollable actions" since all edges in VDTA (that allow output-update) are eager and fired as soon as their constraints become true. But TGA analysis algorithm cannot work for VDTA: edges passing in VDTA cannot be restricted by letting the time progress forever or by changing the inputs. This is because all edges in VDTA are urgent. Moreover, input actions are not explicit in VDTA whereas they are in TGA. Nevertheless, for the analysis of VDTA we have considered the notion of safe time predecessor inspired from [6].

## III. Predecessors of Symbolic States

There are two popular symbolic representations of a set of clock valuations: the region-based [2] and the zone-based [4] representations. Roughly speacking, regions and zones are abstract representations of (infinite) set of valuations; but zones are larger representations: a zone can be decomposed into an equivalent finite set of regions. A region-based analysis algorithm for VDTA inspired by [2] early appeared in [11]. The algorithm is used for selecting real-time test cases for systems modeled with VDTA. The region abstraction allows elegant reasoning on timed systems however it is practically less efficient than the zone abstraction: a single operation over a zone needs many operations over the regions it includes. We consider a zone-based approach for representing symbolic states and to compute their (input/output/time) predecessors.

### A. *Symbolic States, Zones, and Operations*

A symbolic state for $\mathcal{A}$ is a quadruple $(\ell, Z_I, Z_O, Z_X)$ where $\ell \in L$, $Z_X \subseteq \mathbb{R}_+^X$ is the standard *clock-zone* as defined in [4], $Z_I \subseteq Val(I)$ is a subset of $Val(I)$ such that $Z_I = [\![G]\!]$ for some $G \in \mathcal{G}(I)$; the same for $Z_O$. We say that $Z_I$ and $Z_O$ are zones over input and output variables. A symbolic state represents a set of states. We write $(\ell, i, o, x) \in (\ell', Z_I, Z_O, Z_X)$ if and only if $\ell = \ell'$, $i \in Z_I, o \in Z_O$ and $x \in Z_X$. A symbolic state is *stable* iff every state in it is stable.

The input, output and clock-zone $Z_I$, $Z_O$ and $Z_X$ are represented with difference bound matrices (DBM) [4]. DBM are kinds of constraints (comparison between two clocks are allowed). Considering Figure 2, $(\ell_0, L = 1, true, 1 \leq t < 4)$ is one symbolic state where $true$ denotes the constraint that is always true.

**Remark:** zones for input/output values are simpler than the ones used for linear integer systems or other models with variables (like in [1]). This is because constraints in VDTA never compare clock with input/output variables. Abstractions for clock valuations can be separated for abstractions of signal values. It is not the goal of this paper to discuss the effect of this restriction on the expressiveness of the model.

**Remark:** in the sequel, constraints are often considered as zones for simplifying the notations. We implicitly assume that every constraint in $\mathcal{G}^+(I, O, X)$ is equivalent to a set of constraints of $\mathcal{G}(I, O, X)$.

**Computable operations on zones [4]**: if $Z$ is a zone (over clocks or variables), and $A$ is a variable assignment or a clock reset then $[A]Z = \{x \mid x[A] \in Z\}$ denotes the set of valuations from which we can reach a valuation in $Z$ after $A$ is executed. Similarly we define $Z[A] = \{x[A] \mid x \in Z\}$. The *past* of a clock zone $Z$, $Z\!\downarrow = \{x \mid \exists\, \delta \in \mathbb{R}_+, \text{ s.t } x + \delta \in Z\}$ is the set of valuations (the zone) from which valuations in $Z$ can be reached by letting the time elapse.

**Safe time predecessors**: given two clock zones $Z$ and $g$, $Pre_t^s(Z, g)$ denotes the set of *safe time predecessors* [6] of $Z$ with respect to $g$. Intuitively, a clock valuation $x'$ belongs to $Pre_t^s(Z, g)$ if from $x'$ we can reach $x \in Z$ by time elapsing and along the (time) path from $x'$ to $x$ we avoid all valuations in $g$. Later, the safe time predecessor is involved in the computation of the time predecessors. Intuitively, it will prevent the time predecessor to return zones that enable urgent edges. Formally:

$$Pre_t^s(Z, g) = \{x' \in \mathbb{R}_+^X \mid \exists\, \delta \in \mathbb{R}_+\ x \in Z \text{ s.t } x = x' + \delta, \\ \text{and } \forall \delta' \in [0, \delta], \text{ we have } x' + \delta' \notin g\}$$

The computation of $Pre_t^s(Z, g)$ is effective [4], [6]: $Pre_t^s(Z, g) = (Z\!\downarrow \cap\, (\neg(g\!\downarrow))) \cup ((Z \cap (g\!\downarrow) \cap \neg g)\!\downarrow)$ where $Z$, $g$ are convex sets and $\neg g$ is the complement of $g$.

**Partitioning of set of zones**: later, we will also need to split sets of zones into equivalent sets of disjoint zones. Indeed, since zones are abstractions of larger sets of valuations, it could happen that the input/time predecessors of two valuations in a same zone differ according to constraints on urgent edges. So we will need to consider these valuations separately. The operation $Split(\mathcal{C})$ allows to partition a set of sets of valuations (or a set of zones) into a set of *disjoint* sets of valuations (or a set of disjoint zones). Formally, let $\mathcal{C}$ be a finite set of zones (or constraints). $Split(\mathcal{C})$ is a finite set of zones $\{Z^1, \ldots, Z^n\}$ such that: it partitions the set $\mathcal{C}$ meaning that $\bigcup_{i=1..n} Z^i = \bigcup_{Z \in \mathcal{C}} Z$ and $\forall i \neq j\ Z^i \cap Z^j = \emptyset$; and secondly, its elements "match" the clock constraints of $\mathcal{C}$, meaning that $\forall i \in \{1, \ldots, n\}, \forall Z \in \mathcal{C}, Z^i \subseteq Z$ or $Z^i \cap Z = \emptyset$. For example, $Split(\{1 \leq t < 4, 1 < t < 2, 3 < t < 4\}$ may return $\{t = 1, 1 < t < 2, 2 \leq t \leq 3, 3 < t < 4\}$.

### B. *Output-update Predecessors of Symbolic States*

*Definition 5:* The set of output-update predecessors of a symbolic state $(\ell, Z_I, Z_O, Z_X)$, $Pre_o(\ell, Z_I, Z_O, Z_X)$ is defined by:

$$\begin{aligned} Pre_o(\ell, Z_I, Z_O, Z_X) \quad = \quad & \{(\ell', Z_I', Z_O', Z_X') \mid \exists \ell' \xrightarrow{G,A,\mathcal{X}} \ell\ \wedge \\ & Z_I' = Z_I \cap G_I \\ & Z_0' = [A](G_O[A] \cap Z_O) \cap G_O \\ & Z_X' = [\mathcal{X}](G_X[\mathcal{X}] \cap Z_X) \cap G_X\} \end{aligned}$$

*Proposition 1:* $(\ell', i', o', x') \in Pre_o(\ell, Z_I, Z_O, Z_X)$ iff there is $(\ell, i, o, x) \in (\ell, Z_I, Z_O, Z_X)$ such that $(\ell', i', o', x') \in Pre_o(\ell, i, o, x)$

**Proof:**

$\Longrightarrow$ This part of the proof is not difficult.

$\Longleftarrow$ Let $(\ell, i, o, x) \in (\ell, Z_I, Z_O, Z_X)$ then $i \in Z_I$, $o \in Z_O$ and $x \in Z_X$. If $(\ell', i', o', x') \in Pre_o(\ell, i, o, x)$ then $i = i'$, $o = o'[A]$ and $x = x'[\mathcal{X}]$ for some $A \in A(O)$. Moreover, according to the semantics of VDTA there exists an edge $\ell' \xrightarrow{G,A,\mathcal{X}} \ell$ such that $i' \in G_I$, $o' \in G_O$ $x' \in G_X$ and additionally $i = i'$, $o = o'[A]$ and $x = x'[\mathcal{X}]$.

1) we get that $i' \in Z_I \cap G_I$ since $i = i'$, $i' \in G_I$ and $i \in Z_I$.
2) Let us show that $o' \in [A](G_O[A] \cap Z_O) \cap G_O$. Since $o = o'[A]$ and $o' \in Proj_O(G)$ we get that $o \in G_O[A]$. Then $o \in G_O[A] \cap Z_O$ since $o$ also belongs to $Z_O$. Additionally, $o = o'[A]$ implies $o' \in [A]o$. As $o \in G_O[A] \cap Z_O$ we get that $o' \in [A](G_O[A] \cap Z_O)$. Since it is required that $o' \in G_O$ we finally get that

$o' \in [A](G_O[A] \cap Z_O) \cap G_O$.

3) A similar justification is used to show that $x' \in [\mathcal{X}](G_X[\mathcal{X}] \cap Z_X) \cap G_X$.

As there exists an edge $\ell' \xrightarrow{G,A,\mathcal{X}} \ell$, $i' \in Z_I \cap G_I$, $o' \in [A](G_O[A] \cap Z_O) \cap G_O$, and $x' \in [\mathcal{X}](G_X[\mathcal{X}] \cap Z_X) \cap G_X$ we get that $(\ell', i', o', x') \in Pre_o(\ell, Z_I, Z_O, Z_X)$. This ends the proof of the proposition.

## C. Input-update Predecessors of Symbolic States

Let us consider the VDTA in Figure 2 and the symbolic state $s_e = (\ell_0, L = 0, true, 1 \leq t < 4)$. Assume that one wants to compute $Pre_i(s_e)$. As input update transitions keep the outputs and the clocks unchanged, a simple implementation of $Pre_i$ could only replace in $s_e$ the input zone $L = 0$ by the input zone $true$ (i.e, any value of $L$) and return $(\ell_0, true, true, 1 \leq t < 4)$. Such a simple implementation is not correct since it is not possible to execute an input update transition (the one that sets $L$ to 0) from $(\ell_0, 0, 1, 1.5) \in (\ell_0, true, true, 1 \leq t < 4)$ in order to reach $s_e$. Indeed $(\ell_0, 0, 1, 1.5)$ is not stable and the edge $\ell_0 \xrightarrow{L=0;1<t<2;S:=1} \ell_1$ is urgently taken. A correct implementation returns stable symbolic states only. On the other hand, a correct implementation may return the following input predecessors for $s_e$: $(\ell_0, true, true, t = 1), (\ell_0, L \neq 0, true, 1 < t < 2), (\ell_0, true, true, 2 \leq t \leq 3), (\ell_0, L \neq 0, true, 3 < t < 4)$. Clearly, such an implementation has decomposed the output and clock zones of $s_e$ in order to allow/forbid some input values. The decomposition considers the constraints on the outgoing edge of $\ell_0$.

Given a location $\ell$, an output and a clock zone $Z_O$ and $Z_X$ we consider the following sets.
The set of constraints on the outgoing edges of $\ell$ that may become true upon an input update is $Gds(\ell, Z_O, Z_X) = \{G \mid \exists \ell \xrightarrow{G,A,\mathcal{X}} \ell' : G_O \cap Z_O \neq \emptyset$ and $G_X \cap Z_X \neq \emptyset\}$.
*The timing context* of the tuple $(\ell, Z_O, Z_X)$ is the set $Ctx_\delta(\ell, Z_O, Z_X) = \{Z_X\} \cup \{G_X \mid G \in Gds(\ell, Z_O, Z_X)\}$.
*The output context* of the tuple $(\ell, Z_O, Z_X)$ is the set $Ctx_O(\ell, Z_O, Z_X) = \{Z_O\} \cup \{G_O \mid G \in Gds(\ell, Z_O, Z_X)\}$.

Following the example we get that : $Gds(s_e) = \{(L = 1 \wedge 1 < t < 2), (L = 0 \wedge 3 < t < 4)\}$, $Ctx_\delta(\ell_0, true, 1 \leq t < 4) = \{1 \leq t < 4, 1 < t < 2, 3 < t < 4\}$ and $Ctx_O(\ell_0, true, 1 < t < 2) = \{true\}$. The computation of $Pre_i(s_e)$ works as follows: We partition the timing context $Ctx_\delta(\ell_0, true, 1 \leq t < 4)$ into an equivalent set of disjoint "atomic" clock constraints $\{t = 1, 1 < t < 2, 2 \leq t \leq 3, 3 < t < 4\}$. Then, for each atomic clock zone $Z'_X$ in set of disjoint "atomic" clock constraints, we partition $Ctx_O(\ell_0, true, Z'_X)$ into an equivalent set of disjoint "atomic" output constraints. In this example, the result is always $\{true\}$. Finally for each "atomic" clock constraints $Z'_X$ and for each atomic output constraint $Z'_O$ the input predecessor compute the negation of input part of constraints in $Gds(\ell, Z'_O, Z'_X)$.

Let us abstract the above thought into a formal definition.

*Definition 6:* The set of input-update predecessors of a symbolic state $(\ell, Z_I, Z_O, Z_X)$, $Pre_i(\ell, Z_I, Z_O, Z_X)$ is defined by:

$$\begin{aligned} Pre_i(\ell, Z_I, Z_O, Z_X) \;=\; & \{(\ell', Z'_I, Z'_O, Z'_X) \mid \\ & \ell' = \ell, \\ & Z'_X \in Split(Ctx_\delta(\ell, Z_O, Z_X)), \\ & Z'_O \in Split(Ctx_O(\ell, Z_O, Z'_X)), \\ & Z'_I \in \bigcap_{G' \in Gds(\ell, G'_O, Z'_X)} \neg G'_I\} \end{aligned}$$

One can show the following.

*Proposition 2:* $(\ell', i', o', x') \in Pre_i(\ell, Z_I, Z_O, Z_X)$ iff there is $(\ell, i, o, x) \in (\ell, Z_I, Z_O, Z_X)$ such that $(\ell', i', o', x') \in Pre_i(\ell, i, o, x)$

## D. Time Predecessors of Symbolic States

The time predecessor $Pre_t(\ell, Z_I, Z_O, Z_X)$ is implemented analogously to $Pre_i$. Similarly to input updates, the time elapses in stable states only and then we also need to carefully decompose constraints. As far as time elapsing transitions only change the value of clocks, we decompose the input and the output constraints.

We define the sets $Gds(\ell, Z_I, Z_O) = \{G \mid \exists \ell \xrightarrow{G,A,\mathcal{X}} \ell' : G_I \cap Z_I \neq \emptyset$ and $G_O \cap Z_O \neq \emptyset\}$, $Ctx_I(\ell, Z_I, Z_O) = \{Z_I\} \cup \{G_I \mid G \in Gds(\ell, Z_I, Z_O)\}$ and $Ctx_O(\ell, Z_I, Z_O) = \{Z_O\} \cup \{G_O \mid G \in Gds(\ell, Z_I, Z_O)\}$. The set $Gds(\ell, Z_I, Z_O)$ returns the constraints on the outgoing edges of $\ell$ that may become true when one looks back into the time from $(\ell, Z_I, Z_O, Z_X)$.

Consider Figure 2 and assume that one wants to compute $Pre_t(s_2)$ where $s_2 = (\ell_0, L = 0, true, 3 < t < 4)$. Note that although $0 \leq t < 4$ is the past of $3 < t < 4$, the symbolic state $(\ell_0, L = 0, true, 0 \leq t < 4)$ is not a time predecessor of $s_t$ since it is not possible to reach $s_2$ from $(\ell_0, 0, 0, 1.3) \in (\ell_0, L = 0, true, 0 \leq t < 4)$ by letting the time elapse. Indeed, $(\ell_0, 0, 0, 1.3)$ is not stable. Then, we consider $Gds(L = 0, true) = \{L = 0 \wedge 3 < t < 4, L = 0 \wedge 1 < t < 2\}$, $Ctx_I(L = 0, true) = \{L = 0\}$ and $Ctx_O(L = 0, true) = \{true\}$. In a first time one decomposes $Ctx_I(L = 0, true)$ into a set of disjoint "atomic" input constraints and one gets $\{L = 0\}$. Secondly one decomposes $Ctx_O(L = 0, true)$ into a set of disjoint "atomic" output constraints and one gets $\{true\}$. The unique input/output context obtained after the decompositions is given by $L = 0$ and $true$. Then, from the symbolic state $(\ell_0, L = 0, true, 3 < t < 4)$ we compute a clock zone $Z'$ from which we can reach $3 < t < 4$ by time elapsing and along the (time) path from $Z'$ to $3 < t < 4$ we must avoid the clock part of all constraints in $Gds(L = 0, true)$ since otherwise one reach non stable states. This correspond to the computation of *safe time predecessors* of a clock zone with respect to another one. For instance we compute the safe time predecessor of $3 < t < 4$ with respect to $\{3 < t < 4, 1 < t < 2\}$ and we should get $2 \leq t \leq 3$. Finally we should get that $Pre_t(s_2) = (\ell_0, L = 0, true, 2 \leq t \leq 3)$.

For the computation of $Pre_t(\ell, Z_I, Z_O, Z_X)$, the next step after the decomposition of input/output constraints is to compute the *safe time predecessors*. We define the time predecessor as follows:

*Definition 7:* The set of time predecessors of a symbolic state $(\ell, Z_I, Z_O, Z_X)$, $Pre_i(\ell, Z_I, Z_O, Z_X)$ is defined by:

$$\begin{aligned} Pre_t(\ell, Z_I, Z_O, Z_X) = \{(\ell', Z'_I, Z'_O, Z'_X) \mid \\ \ell' = \ell, \\ Z'_I \in Split(Ctx_I(\ell, Z_I, Z_O)), \\ Z'_O \in Split(Ctx_O(\ell, Z'_I, Z_O)), \\ Z'_X \in Pre^s_t(Z_X, \{G_X \mid G \in Gds(\ell, Z'_I, Z'_O)\})\} \end{aligned}$$

One can show the following.

*Proposition 3:* $(\ell', i', o', x') \in Pre_t(\ell, Z_I, Z_O, Z_X)$ iff there is $(\ell, i, o, x) \in (\ell, Z_I, Z_O, Z_X)$ such that $(\ell', i', o', x') \in Pre_t(\ell, i, o, x)$

### E. Zone-Based Reachability Analysis Algorithm

The zone based reachability analysis algorithm computes at each step the predecessors of already encountered symbolic states. The predecessors are computed as follows:

$$Pre(\ell, Z_I, Z_O, Z_X) = \bigcup_{\tau \in \{i,o,t\}} Pre_\tau(\ell, Z_I, Z_O, Z_X)$$

Note that all the operations on zones (update, pre-update, past time, $Split$) can be effectively computed [4] using DBM (or simple adaptations for input and output zones). The iterative fixpoint process of Algorithm 1 converges after finite many steps. This is true as the computation of $Pre_i$, $Pre_o$, $Pre^s_t$ and $Pre_t$ involves exactly the constants in $\mathcal{A}$ and variable updates only set variables to constants. Using similar argument to [6], one can show that the reachability analysis algorithm is EXPTIME. Although we did not carry out experiments of the zone-based algorithm, it should be clear that this new algorithm for VDTA is practically more efficient than that based on regions [11].

## IV. Conclusion and Future Work

The paper proposed a more efficient zone-based backward algorithm for the reachability analysis of Variable Driven Timed Automata (VDTA). The algorithm is based on the definition of predecessors operators of symbolic states. Symbolic states contain input, output, and clock constraints that should be respected in order to reach the target state. These informations can be used by the environment to control the execution of systems or select test cases.

The VDTA model received a favorable echo among some industrial partners since it is a good compromise between simplicity and expressiveness. Ongoing works include composability issues. Moreover, one can intensify the model with tough arithmetic operations and study decidability issues.

## Acknowledgment

This work has been supported by the ANR (Agence Nationale de la Recherche) TesTEC Project. We also would like to thank Hervé Marchand for his comments.

## References

[1] Rajeev Alur, Thao Dang, and Franjo Ivancic. Reachability analysis of hybrid systems via predicate abstraction. In *5th International Workshop on Hybrid Systems: Computation and Control (HSCC'02)*, pages 35–48. Springer, 2002.

[2] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] Roberto Barbuti and Luca Tesei. Timed automata with urgent transitions. *Acta Informatica*, 40(5), 2004.

[4] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithm and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124. Springer, 2004.

[5] Patricia Bouyer. Untameable timed automata! In *20th Symposium on Theoretical Aspects of Computer Science (STACS'03)*, pages 620–631. Springer, 2003.

[6] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *16th International Conference on Concurrency Theory (CONCUR'05)*, pages 66–80. Springer, 2005.

[7] Frédéric Herbreteau and B. Srivathsan. Efficient on-the-fly emptiness check for timed büchi automata. In *8th Int. Symp. on Automated Technology for Verification and Analysis (ATVA'10)*, pages 218–232. Springer, 2010.

[8] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. Timed i/o automata: A mathematical framework for modeling and analyzing real-time systems. In *The 24th IEEE International Real-Time Systems Symposium (RTSS'03)*, volume 0, pages 166–177. IEEE, 2003.

[9] Stephan Merz. Model checking: A tutorial overview. In *Modeling and Verification of Parallel Processes*, pages 3–38. Springer, 2001.

[10] Houda Bel Mokadem, Béatrice Bérard, Patricia Bouyer, and François Laroussinie. A new modality for almost everywhere properties in timed automata. In *16th International Conference on Concurrency Theory (CONCUR'05)*, pages 110–124. Springer, 2005.

[11] Omer Nguena-Timo, Hervé Marchand, and Antoine Rollet. Automatic test generation for data-flow reactive systems with time constraints. In *22nd IFIP International Conference on Testing Software and Systems: Short Papers (ICTSS'10 )*. CRIM-Canada, 2010.

[12] Omer Nguena-Timo and Antoine Rollet. Conformance testing of variable driven automata. In *8th IEEE International Workshop on Factory Communication Systems Communication in Automation (WFCS'10)*, 2010.

[13] Omer Nguena-Timo and Antoine Rollet. Test selection for data-flow reactive systems based on observations. In *Software Testing, Verification and Validation Workshops (ICSTW'11)*. IEEE Xplore, 2011.

[14] Peter J. Ramadge and W. Murray Wohnam. The control of discrete event systems. *Proceedings of the IEEE Computer Society*, 77:81–98, 1989.

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