Off-line Test Case Generation For Timed Symbolic Model-Based Conformance Testing[⋆]

Boutheina Bannour[1], Jose Pablo Escobedo[2], Christophe Gaston[2] and Pascale Le Gall[3]

[1] Sherpa Engineering, 92250, La Garenne Colombes, France
email: b.bannour@sherpa-eng.com
[2] CEA LIST, Point Courrier 174, 91191, Gif-sur-Yvette, France
email: {jose-pablo.escobedo, christophe.gaston}@cea.fr
[3] Laboratoire MAS, Grande Voie des Vignes, 92195 Châtenay-Malabry, France
email: pascale.legall@ecp.fr

Abstract. Model-based conformance testing of reactive systems consists in taking benefit from the model for mechanizing both test data generation and verdicts computation. On-line test case generation allows one to apply adaptive on-the-fly analyzes to generate the next inputs to be sent and to decide if observed outputs meet intended behaviors. On the other hand, in off-line approaches, test suites are pre-computed from the model and stored under a format that can be later performed on test-beds. In this paper, we propose a two-passes off-line approach where: for the submission part, a test suite is a simple timed sequence of numerical input data and waiting delays, and then, the timed sequence of output data is post-processed on the model to deliver a verdict. As our models are Timed Output Input Symbolic Transition Systems, our off-line algorithms involve symbolic execution and constraint solving techniques.

Keywords: Model-based testing, off-line testing, real-time systems, test suite generation, verdict computation, symbolic execution, timed output-input symbolic transition systems.

1 Introduction

Using formal methods to generate test cases and to compute verdicts has been widely studied in the frame of Model Based Testing. In the domain of reactive systems, models are often given as *labeled transition systems* which describe the expected sequences of input and output data (called *traces*). Real executions of the System Under Test (SUT) can also be seen as traces. Testing an SUT comes to interact with it to build traces which are analyzed regarding to its model to provide verdicts. In black box testing, an SUT is often hardly controllable at the test execution phase, typically because, for the sake of abstraction, its reference model may include non-deterministic situations (*i.e.* after a given trace, several outputs may occur). For this reason, when dealing with automatic test case generation, approaches in which inputs to be sent to the SUT are computed

[⋆] Work partially supported by the ITEA project openETCS.

on-the-fly are very popular: they permit to stimulate it in a flexible manner depending on observed SUT executions, and depending on the goal of the testing process in terms of behaviors to cover. Such approaches are often qualified as *on-line testing*.

The other alternative consists in: computing the full input sequence; submitting the sequence to the SUT; storing the output sequence of the SUT during the execution phase; computing *a posteriori* a verdict by analyzing the trace resulting from the merge of input and output sequences. Such approaches, qualified as *off-line testing* ones, have several advantages. First, computed input sequences can be stored and later translated into several formats, in particular to become compatible with various home made test benches in different industrial contexts. This allows one to avoid the intertwining (unavoidable in on-line approaches) of the test generation/test execution/verdict computation processes, which may be technically hard to achieve. Second, tests can be replayed as many times as desired which makes off-line methods particularly well-adapted for non-regression testing. Third, by construction, no constraint solving delays can interfere with the test data execution. To sum up, off-line testing eases the deployment of input sequences in the test environment and enables their reuse.

However, a particular source of concern about off-line testing is to know at which instants precisely the tester has to send the successive data of the input sequence. Those instants can be identified from the knowledge of a clock cycle or of an hypothesis of an instantaneous reaction in synchronous frameworks (e.g. testing from the clocked data-flow language Lustre in [10] or from Finite State Machines –FSM– in [11]). In asynchronous systems, the waiting delay between two successive inputs is important because the SUT takes time to compute outputs to be sent, and because sending an input before the output computation is complete, or after it is completed, leads to define different execution traces. Therefore, timed models introducing time delays between communication actions are good candidates to support input sequence generation. Moreover, in order to properly identify the trace observed at the test execution step, one needs a mechanism to know the order of occurrences between inputs and outputs: in fact, measuring delays between outputs permits to reconstruct the full trace of inputs, outputs and delays.

In this paper, we propose an off-line testing approach in a timed model-based framework. The approach is decoupled in different steps: (a) the coverage-based selection of an input sequence with delays; (b) the execution of the SUT for the input sequence which generates an output sequence with delays as a result. Input and output sequences are then merged to generate a complete execution trace; (c) the verdict computation based on a traversal of the model guided by the execution trace. The first and last steps are conducted off-line with Timed Input Output Symbolic Transition Systems (TIOSTS) for models. TIOSTS are extensions of Input/Output Symbolic Transition Systems (IOSTS) [8] and of Timed Automata (TA) [1], in which both data and time properties are expressed symbolically. Our framework is situated within the context of the *tioco* conformance relation [5, 9, 14].

Section 2 gives preliminaries about time and data denotation as first order structures. In Section 3, we recall the *tioco* setting. Then, we present the syntax of TIOSTS and give their semantics as timed traces in Section 4. We show in the same section how to compute these traces using symbolic execution techniques. In Section 5, we introduce our off-line testing algorithm. Section 6 reviews relevant state of the art concerning timed conformance testing.

2 Data and Time Denotation

We use classical multi-typed first order logic to symbolically denote data and time. A *signature* Ω is a triple (S, Op, P) where S is a set of *types*, Op (resp. P) is a set of *operations* (resp. *predicates*) provided with a profile in S^+ (resp. S^*). For any set V of variables typed in S, we note $T_\Omega(V)$ (resp. $P_\Omega(V)$) the set of *terms* (resp. *predicate terms*) over V and Ω inductively defined as usual. An *Ω-model* $M = \bigcup_{s \in S} M_s$ is provided with a function $\overline{f} : M_{s_1} \times \cdots \times M_{s_n} \to M_s$ (resp. a predicate $\overline{p} : M_{s_1} \times \cdots \times M_{s_n}$) for each $f : s_1 \cdots s_n \ \to \ s$ in Op (resp. for each $p : s_1 \cdots s_n$ in P). *Substitutions* (resp *interpretations*) are applications from V to $T_\Omega(V)$ (resp. M) preserving types and can be canonically extended to $T_\Omega(V)$.[4] The set $Sen_\Omega(V)$ of all *formulas* contains the predicate terms (including the truth values $\top$ and $\bot$ denoting resp. the true and false values), the equalities $t = t'$ for t, t' terms of the same type and all formulas built over the usual connectives $\neg, \vee, \wedge$ and quantifiers $\forall x$, $\exists x$ with x variable of V.

The satisfaction of a formula φ by an interpretation $\nu : V \to M$ is denoted $M \models_\nu \varphi$ where $M \models_\nu t = t'$ (resp. $M \models_\nu p(t_1, \ldots, t_n)$ with $t_1, \ldots, t_n$ terms of $T_\Omega(V)$) is defined by $\nu(t) = \nu(t')$ (resp. $(\nu(t_1), \ldots, \nu(t_n)) \in \overline{p}$), and connectives and quantifiers are handled as usual in more complex formulas.

We suppose that Ω contains a particular type *time* to denote durations. For readability sake, M_{time} is denoted D (for *Duration*) and is assimilated to the set of positive (or null) real numbers.[5] Op and P contain some classical operations $+ : time \times time \to time$, $- : time \times time \to time$, or predicates $<, \leq : time \times time$, provided by default with their usual meanings.[6]. Variables of type *time* are called *clocks* For a set of clocks T, we note $Sen_{time}(T)$ the set of formulas only containing conjunctions of formulas of the form $z \leq d$, $d \leq z$, $z < d$ or $d < z$, where d is a constant and z is in T.

In the sequel, $\Omega = (S, Op, P)$ and M are supposed given.

3 System Under Test

In order to reason about Systems Under Test, we denote them as Timed Input Output Labeled Transition Systems (TIOLTS) [6, 9, 14]. TIOLTS are automata

[4] The set of applications from A to B is denoted as B^A.

[5] In practice, any set of values used in a constraint solver for approaching real numbers.

[6] For simplicity, $\overline{+}$, $\overline{<}$ . . . are also denoted by $+$, $<$.

whose transitions are labeled either by actions (inputs, outputs) or by delays. For simplicity, the unobservable action τ is not introduced (see [2]).

Let C be a set of *channels*. The *set of actions over* C, denoted $Act_M(C)$, is $I_M(C) \cup O_M(C)$ where $I_M(C) = \{c?v \mid v \in M, c \in C\}$ denotes the set of *inputs* and $O_M(C) = \{c!v \mid v \in M, c \in C\}$ denotes the set of *outputs*. Thus, $c?v$ (resp. $c!v$) stands for the reception (resp. emission) of v by the SUT on the channel c.

Definition 1 (*TIOLTS*)**.** *A TIOLTS* over C *is a triple* (Q, q_0, Tr) *where* Q *is a set of* states*,* $q_0 \in Q$ *is the* initial state*, and* $Tr \subseteq Q \times (Act_M(C) \cup D) \times Q$ *is a set of* transitions.

For any $tr = (q, a, q')$ of Tr, $source(tr)$, $act(tr)$, and $target(tr)$ stand respectively for q, a, and q'. The set of *paths* of a TIOLTS $\mathbb{A} = (Q, q_0, Tr)$ is the set $Path(\mathbb{A}) \subseteq Tr^*$ containing the empty sequence ε and all sequences $tr_1 \dots tr_n$ such that $source(tr_1) = q_0$, and for all $i < n, target(tr_i) = source(tr_{i+1})$.[7] Let p be a path of $\mathbb{A}$, the *trace of* p, denoted as $trace(p)$, is ε if $p = \varepsilon$, $trace(p')$ if $p = p'.tr$ with $act(tr) = 0$, and $act(tr).trace(p')$ if $p = p'.tr$ with $act(tr) \neq 0$. $Traces(\mathbb{A})$ is the set of traces of all paths of $Path(\mathbb{A})$. The set $TTraces(\mathbb{A})$ of *timed traces of* $\mathbb{A}$ is the smallest set containing $Traces(\mathbb{A})$ and such that:

- for any $\sigma = \sigma'.d.\sigma''$ in $TTraces(\mathbb{A})$, $\sigma'.d_1.d_2.\sigma''$ is in $TTraces(\mathbb{A})$,
- for any $\sigma = \sigma'.d_1.d_2.\sigma''$ in $TTraces(\mathbb{A})$, $\sigma'.d.\sigma''$ is in $TTraces(\mathbb{A})$,
- for any $\sigma.r$ in $TTraces(\mathbb{A})$ with r in $Act_M(C) \cup (D \backslash \{0\})$, σ is in $TTraces(\mathbb{A})$,

where d, d_1 and d_2 are any delays of $D \backslash \{0\}$ verifying $d = d_1 + d_2$.

We introduce a normalization operation whose purpose is to compute a trace in which the occurring delays are the largest possible (by adding all consecutive delays of a trace). Let σ be a trace of $(Act_M(C) \cup (D \backslash \{0\}))^*$, $\overline{\sigma}$ is ε if $\sigma = \varepsilon$, $\overline{\sigma'}.a$ if $\sigma = \sigma'.a$ with $a \in Act_M(C)$, and $\overline{\sigma'}.(d_1 + \cdots + d_n)$ if $\sigma = \sigma'.d_1 \cdots d_n$ where for all $i \leq n$, $d_i \in D \setminus \{0\}$, and σ' is either ε or terminated by an action in $Act_M(C)$.

We naturally define the trace duration as the sum of all delays occurring in a trace. More precisely, for a trace σ, $duration(\sigma)$ is 0 if $\sigma = \varepsilon$, $duration(\sigma')$ if $\sigma = \sigma'.r$ with $r \in Act_M(C)$, and $duration(\sigma') + d$ if $\sigma = \sigma'.d$ with $d \in D \setminus \{0\}$.

Definition 2 (*SUT*)**.** *A SUT* over C *is a TIOLTS* $\mathbb{S} = (Q, q_0, Tr)$ *over* C *satisfying the following properties:*

- **Input enableness***:* $\forall\ q \in Q$*,* $c \in C$*,* $v \in M$*, there exists* $(q, c?v, q')$ *in* Tr*,*
- **Time elapsing***:* $\forall q \in Q$ *s.t. there is no transition of the form* $(q, c!v, q')$ *in* Tr*, then there exists* (q, d, q') *in* Tr *with* d *in* $D \backslash \{0\}$.

Input enableness condition is very classical: it expresses that an SUT cannot refuse an input. **Time elapsing** condition expresses that the absence of a reaction amounts to observe no reaction during a strictly positive delay.

The conformance relation *tioco* [5, 7, 9, 14] defines the correctness of an SUT w.r.t a TIOLTS model.

[7] A^* is the set of words on A with ε as the empty word and "." as the concatenation law.

Definition 3 (tioco). *Let $\mathbb{S}$ be an SUT and $\mathbb{A}$ a TIOLTS, both defined over C.* $\mathbb{S}$ conforms to $\mathbb{A}$*, denoted $\mathbb{S}$ tioco $\mathbb{A}$, if and only if for any σ in $TTraces(\mathbb{A})$ and r in $O_M(C) \cup (D\backslash\{0\})$ we have:*

$$\sigma.r \in TTraces(\mathbb{S}) \Longrightarrow \sigma.r \in TTraces(\mathbb{A})$$

In the Introduction, we argued that test data, in off-line testing, is made of a test input sequence to be submitted to the SUT and of a test output sequence produced by the SUT. A natural testing hypothesis expresses that when these two sequences are grouped, they form a trace of the SUT. In order to make the connection between test data and the SUT, we start by introducing some functions to handle traces: the projection function allows us to extract a sub-trace, and the merge allows to combine two traces according to delays occurring in them. Formally, for any trace σ, the *input projection of* σ, denoted $\sigma_{\downarrow_I}$, is: ε if $\sigma = \varepsilon$; $\sigma'_{\downarrow_I}$ if $\sigma = \sigma'.o$ with $o \in O_M(C)$, and $\sigma'_{\downarrow_I}.x$ if $\sigma = \sigma'.x$ with $x \in I_M(C) \cup (D\backslash\{0\})$. Similarly, the *output projection of* σ, denoted $\sigma_{\downarrow_O}$, is defined by exchanging the roles of inputs and outputs. Let us define the merge operation by induction on the trace structure. For that purpose, let us consider two traces σ_i in $(I_M(C) \cup (D\backslash\{0\}))^*$ and σ_o in $(O_M(C) \cup (D\backslash\{0\}))^*$ defined over C. $Merge(\sigma_i, \sigma_o)$ is defined as follows:

- σ_o (resp. σ_i) if $\sigma_i = \varepsilon$ (resp. $\sigma_o = \varepsilon$),
- $o.Merge(\sigma_i, \sigma')$ if $\sigma_o = o.\sigma'$ with $o \in O_M(C)$,
- $i.Merge(\sigma', \sigma_o)$ if $\sigma_i = i.\sigma'$ with $i \in I_M(C)$ and $\sigma_o = d.\sigma'$ with $d \in D\backslash\{0\}$,
- with $\sigma_i = d_i.\sigma'_i$, $d_i \in D\backslash\{0\}$, and $\sigma_o = d_o.\sigma'_o$, $d_o \in D\backslash\{0\}$
  - $d_i.Merge(\sigma'_i, (d_o - d_i).\sigma'_o)$ if $d_i < d_o$
  - $d_i.Merge(\sigma'_i, \sigma'_o)$ if $d_i = d_o$
  - $d_o.Merge((d_i - d_o).\sigma'_i, \sigma'_o)$ if $d_o < d_i$.

For merging two traces beginning with an input i and then an output o, we choose to prioritize the output o: if from the point of view of the tester, i and o are perceived as occurring at the same time, it is likely that o follows from the previous inputs of the input sequence, not from i. Thus, placing o before i in the merging trace explicits that i cannot be a cause of o.

As already explained, off-line test input sequences are modeled as sequences of inputs and strictly positive delays.

Definition 4 (input sequence). *An* input sequence *over C is a sequence of $(I_M(C) \cup (D\backslash\{0\}))^*$.*

Once an input sequence is considered, the test execution phase amounts to play it on $\mathbb{S}$ so that it produces an output sequence σ_o made of outputs and strictly positive delays. Modeling $\mathbb{S}$ as a TIOLTS leads to the following facts:

• there exists a trace σ of $TTraces(\mathbb{S})$, such that the input (resp. output) projection of σ corresponds to the submitted input sequence (resp. the output sequence collected when executing the input sequence);

• the duration of the test output sequence is strictly greater than the one of the input sequence. Indeed, when collecting the output sequence, the tester will at least wait a moment after sending the last input to the SUT.

Definition 5 (execution). *Let $\mathbb{S}$ be an SUT over C, and let σ_i be an input sequence over C.*

An execution *of σ_i on $\mathbb{S}$, is defined as a sequence σ_o of $(O_M(C) \cup (D \backslash \{0\}))^*$ verifying:* (1) $Merge(\sigma_i, \sigma_o) \in TTraces(\mathbb{S})$ *and* (2) $duration(\sigma_o) > duration(\sigma_i)$.

σ_o *is called an* output sequence *of σ_i for $\mathbb{S}$ and we note $\sigma_i \rightsquigarrow_{\mathbb{S}} \sigma_o$.*

The condition (2) ensures that the trace $Merge(\sigma_i, \sigma_o)$ is terminated by either an output or a delay, as verified on the example of Figure 1. Moreover, let us point out that due to non-determinism, there can exist two distinct traces σ_o and σ_o' such that $\sigma_i \rightsquigarrow_{\mathbb{S}} \sigma_o$ and $\sigma_i \rightsquigarrow_{\mathbb{S}} \sigma_o'$.

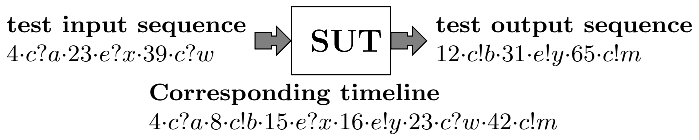
test input sequence
$4{\cdot}c?a{\cdot}23{\cdot}e?x{\cdot}39{\cdot}c?w$
SUT
test output sequence
$12{\cdot}c!b{\cdot}31{\cdot}e!y{\cdot}65{\cdot}c!m$
Corresponding timeline
$4{\cdot}c?a{\cdot}8{\cdot}c!b{\cdot}15{\cdot}e?x{\cdot}16{\cdot}e!y{\cdot}23{\cdot}c?w{\cdot}42{\cdot}c!m$

Fig. 1: Example of merging two test traces.

Timed Input Output Symbolic Transition Systems (TIOSTS) are models where data and time are symbolically specified. They are well-accepted concise representations of TIOLTS: symbolic data allow one to characterize internal states, to express firing conditions of transitions and to denote exchanged messages, while real-time properties are handled with constraints and resets on clock variables.

Since *tioco* defines conformance of an SUT w.r.t. a TIOLTS specification, TIOSTS model semantics (Definition 8) is given in the form of TIOLTS.

4 Timed Input Output Transition Systems

TIOSTS are defined over a signature $\Sigma = (C, A, T)$ where C is a set of channels, A is a set of variables (whose type is not *time*) called *attribute variables*, and T is a set of clocks. The set of symbolic actions $Act(\Sigma)$ is $I(\Sigma) \cup O(\Sigma)$ with $I(\Sigma) = \{c?x | x \in A, c \in C\}$ and $O(\Sigma) = \{c!t | t \in T_\Omega(A), c \in C\}$.

Definition 6 (TIOSTS). *A TIOSTS over Σ is a triple (Q, q_0, Tr), where Q is a set of* states*, $q_0 \in Q$ is the* initial state *and Tr is a set of* transitions *of the form $(q, \phi, \psi, \mathbb{T}, act, \rho, q')$ where $q, q' \in Q$, $\phi \in Sen_{time}(T)$, $\psi \in Sen_\Omega(A)$, $\mathbb{T} \subseteq T$, $act \in Act(\Sigma)$, and $\rho : A \to T_\Omega(A)$ is a substitution s.t. x does not occur in ψ if act is of the form $c?x$.*

When firing a transition $(q, \phi, \psi, \mathbb{T}, act, \rho, q')$, ϕ is a formula constraining the delay at which the action *act* occurs, ψ is a firing condition on attribute variables, ρ assigns new values to attribute variables, and clocks in $\mathbb{T}$ are reset. The restriction about the occurrence of the reception variable x in the firing condition ψ is due to the fact that both formulas ϕ and ψ are evaluated precisely at the reception instant.

Example 1 (Trajectory module of a Flight Management System).
The *Trajectory* module is embedded in a plane and orchestrates the computation of plane trajectories. Figure 2 depicts its specification: it waits to receive the current location of the aircraft (transition $q_0 \to q_1$); it sends a request to access the flight plan in less than 1 time units (transition $q_1 \to q_2$); then, either it receives the requested flight plan in less than 2 time units (transition $q_2 \to q_3$) or it does not receive it and sends an error message to its environment in less than 3 time units (transition $q_2 \to q_0$); after this step, the trajectory module sends a request for parameters (typically, fuel quantity, speed, etc.) related to the state of the plane (transition $q_3 \to q_4$), and again either receives them in less than 1 time unit or sends a warning message in less than 2 time units (two transitions $q_4 \to q_5$); then, the module sends in less than 1 time unit the location, the flight plan and the parameters to a calculator (transitions $q_5 \to q_6$), which replies by sending the new trajectory of the plane in less than 2 time units (transition $q_6 \to q_7$) unless it fails to meet the time constraint, in which case the module sends an error message in less than 3 time units; if the calculator succeeds to react on time, the computed navigation commands are transmitted to the environment (transition $q_7 \to q_0$) in a total of less than 9 time units.

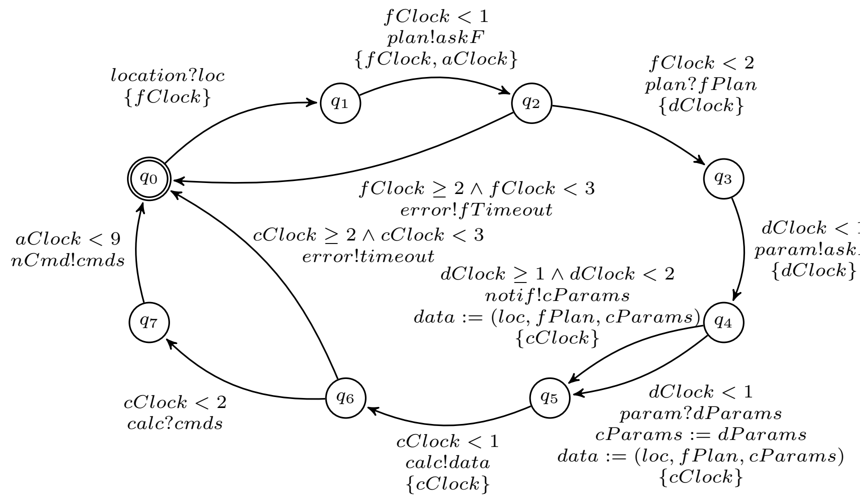

Fig. 2: TIOSTS for the Trajectory module.

Executions of TIOSTS transitions are called *runs* and modeled as TIOLTS transitions whose states are called *snapshots*:

Definition 7 (runs of transitions). *Let $\mathbb{G} = (Q, q_0, Tr)$ be a TIOSTS over Σ. The set $Snp_M(\mathbb{G})$ of* snapshots *of $\mathbb{G}$ is the set $Q \times D \times M^{A \cup T}$. For any $tr = (q, \phi, \psi, \mathbb{T}, act, \rho, q') \in Tr$, the* set of runs of *tr is the set $Run(tr) \subseteq Snp_M(\mathbb{G}) \times Act_M(C) \times Snp_M(\mathbb{G})$ s.t. $((q, \mathcal{T}, \nu), act_M, (q', \mathcal{T}', \nu')) \in Run(tr)$ iff there exist $d \in D$ and $\nu^i : A \cup T \to M$ satisfying:*

• $\mathcal{T}' = \mathcal{T} + d$,
• *for all* $w \in T$, $\nu^i(w) = \nu(w) + d$,
• *if* $act = c!t$ *then for all* $x \in A$, $\nu^i(x) = \nu(x)$,
• *if* $act = c?x$ *then for all* $y \in A \setminus \{x\}$, $\nu^i(y) = \nu(y)$,
such that we have: $act_M = \nu^i(act)$, $\forall x \in A, \nu'(x) = \nu^i(\rho(x))$, $\forall w \in \mathbb{T}$, $\nu'(w) = 0$, $\forall w \in (T \setminus \mathbb{T}), \nu'(w) = \nu^i(w)$, $M \models_{\nu_i} \phi$ *and* $M \models_{\nu_i} \psi$.

Based on runs of transitions, we associate a TIOLTS to a TIOSTS:

Definition 8 (TIOLTS associated to a TIOSTS). *The* TIOLTS over C associated to $\mathbb{G}$, *denoted* $LTS_{\mathbb{G}} = (Snp_M(\mathbb{G}) \cup \{init, q_\delta\}, init, Tr')$, *is defined as follows:* $init, q_\delta$ *are two distinct states not belonging to* $Snp_M(\mathbb{G})$ *and* Tr' *is the smallest subset of* $(Snp_M(\mathbb{G}) \cup \{init, q_\delta\}) \times (Act_M(C) \cup D) \times (Snp_M(\mathbb{G}) \cup \{q_\delta\})$ *s.t.*
Initialization*: for any* $((q_0, 0, \nu_0), act_M, (q, \mathcal{T}, \nu)) \in Run(tr)$ *with* $tr \in Tr$ *and* $\forall w \in T, \nu_0(w) = 0$,
• *if* $0 < \mathcal{T}$, $(init, \mathcal{T}, (q_0, \mathcal{T}, \nu_0))$ *and* $((q_0, \mathcal{T}, \nu_0), act_M, (q, \mathcal{T}, \nu))$ *are in* Tr',
• *else* $(\mathcal{T} = 0)$ $((q_0, 0, \nu_0), act_M, (q, 0, \nu))$ *is in* Tr',
Runs*: for* $((q, \mathcal{T}, \nu), act_M, (q', \mathcal{T}', \nu')) \in Run(tr)$ *with* $tr \in Tr$ *and* $q \neq q_0$,
• *if* $\mathcal{T} < \mathcal{T}'$, $((q, \mathcal{T}, \nu), \mathcal{T}' - \mathcal{T}, (q, \mathcal{T}', \nu))$, $((q, \mathcal{T}', \nu), act_M, (q', \mathcal{T}', \nu'))$ *are in* Tr',
• *else* $(\mathcal{T} = \mathcal{T}')$ $((q, \mathcal{T}, \nu), act_M, (q', \mathcal{T}, \nu'))$ *is in* Tr',
Quiescence*: let snp in* $Snp_M(\mathbb{G}) \cup \{init\}$ *be a snapshot s.t. there does not exist* $(snp, act_M, snp') \in Tr'$ *with* $act_M \in O_M(C) \cup (D \setminus \{0\})$, *then* $(snp, d, q_\delta) \in Tr'$ *for any* $d \in D \setminus \{0\}$.

Initialization transitions are introduced to consider any possible interpretation of attribute variables at the beginning of executions (clocks are set to 0), **Runs** transitions naturally correspond to the execution of TIOSTS transitions, and **Quiescence** transitions state that if no reaction (output or delay transitions) is specified at a given state, then waiting for any delay from this state is possible. We note $TTraces(\mathbb{G})$ for $TTraces(LTS_{\mathbb{G}})$.

In the sequel, we will reason about traces of $TTraces(\mathbb{G})$ by using symbolic execution techniques. They consist in: executing the TIOSTS for symbolic values rather than numerical ones, and computing constraints on those values for all possible TIOSTS executions. In order to represent symbolic values, we suppose that a set of fresh variables $F = \bigcup_{s \in S} F_s$ is given. *Symbolic states* are structures used to store pieces of information concerning an execution:

Definition 9 (symbolic state). *A* symbolic state for $\mathbb{G}$ *is a tuple* $(q, \theta, \pi, \vartheta, \lambda)$ *where* $q \in Q$, $\theta \in Sen_{time}(F_{time})$, $\pi \in Sen_{\Omega}(F)$, $\vartheta \in T_{\Omega}(F_{time})$ *and* $\lambda : A \cup T \to T_{\Omega}(F)$ *is an application preserving types.*

We note $\mathcal{S}$ *the set of all symbolic states over* F.

For a symbolic state $\eta = (q, \theta, \pi, \vartheta, \lambda)$, q denotes the state reached after an execution leading to η, θ is a constraint on symbolic delay values called *time path condition*, π is a constraint on symbolic data values called *data path condition*, ϑ denotes the duration from the beginning of the execution leading to η, and λ denotes terms over symbolic variables in F that are assigned to variables of A.

In the sequel, Σ_F stands for (F, C). Moreover for any symbolic state $\eta = (q, \theta, \pi, \vartheta, \lambda)$, $q(\eta)$, $\theta(\eta)$,$\pi(\eta)$, $\vartheta(\eta)$ and $\lambda(\eta)$ stand resp. for q, θ, π, ϑ and λ. For an application $\lambda : A \cup T \to T_\Omega(F)$, we extend it in a canonical way to $T_\Omega(A \cup T)$, $Sen_\Omega(A \cup T)$ and $Act(\Sigma)$. All these extensions are also simply denoted by λ. The symbolic execution of a TIOSTS is based on symbolic executions of transitions:

Definition 10 (symbolic execution of transitions). *Let $\mathbb{G} = (Q, q_0, Tr)$ be a TIOSTS over Σ and $tr = (q, \phi, \psi, \mathbb{T}, act, \rho, q')$ be in Tr. A* symbolic execution *of tr from η in $\mathcal{S}$ is $st = (\eta, act_F, \eta') \in \mathcal{S} \times Act(\Sigma_F) \times \mathcal{S}$ such that $q(\eta') = q'$, $\vartheta(\eta') = \vartheta(\eta) + z$ where $z \in F_{time}$ is a new fresh variable, and there exists $\lambda^i : A \cup T \to T_\Omega(F)$ satisfying:*
- *if $act \in O(\Sigma)$, then that for all $x \in A$, $\lambda^i(x) = \lambda(x)$,*
- *if $act \in I(\Sigma)$ of the form $c?y$ then $\lambda^i(y)$ is a fresh variable of F and for all $x \in A \setminus \{y\}$, $\lambda^i(x) = \lambda(x)$,*
- *for all $w \in T$ we have $\lambda^i(w) = \lambda(w) + z$,*

such that $act_F = \lambda^i(act)$, for all $x \in A$, $\lambda(\eta')(x) = \lambda^i(\rho(x))$, for all $w \in (T \setminus \mathbb{T})$, $\lambda(\eta')(w) = \lambda^i(w)$, for all $w \in \mathbb{T}$, $\lambda(\eta')(w) = 0$, Finally $\pi(\eta') = \pi(\eta) \wedge \lambda^i(\psi)$ and $\theta(\eta') = \theta(\eta) \wedge \lambda^i(\phi)$.

The variable z is called the *delay of st* and is denoted $delay(st)$. $source(st)$, $act(st)$ and $target(st)$ stand respectively for η, act_F and η'. In the sequel, for any symbolic transition $st = (\eta, act_F, \eta')$, we note $Fresh(st) = \{delay(st)\}$ if $act \in O(\Sigma)$ and $Fresh(st) = \{delay(st), \lambda^i(y)\}$ if $act = c?y$. The symbolic execution tree associated to the TIOSTS is then defined as follows:

Definition 11 (symbolic execution of a TIOSTS). *A symbolic execution of $\mathbb{G} = (Q, q_0, Tr)$ is a couple $SE(\mathbb{G}) = (Init, ST)$ where:*
- *$Init = (q_0, \top, \top, 0, \lambda_0)$ is such that $\forall x \in T, \lambda_0(x) = 0$ and for all distinct variables x, y in A, $\lambda_0(x)$ and $\lambda_0(y)$ are distinct variables of F,*
- *ST is the set of all symbolic executions st of tr in Tr from $\eta \in \mathcal{S}$ with $q(\eta) = source(tr)$ and $Fresh(st) \cap \lambda_0(A) = \emptyset$. Moreover for any two distinct $st_1, st_2 \in ST$, $Fresh(st_1) \cap Fresh(st_2) = \emptyset$.*

Example 2 (Symbolic execution). Figure 3 depicts the symbolic execution of the Trajectory module of Example 1. For the sake of clarity: only one path of the tree is shown, representing a complete cycle of the *Trajectory* module, transitions deviating from this path are cut; we show the delay of the transition together with its symbolic action; and, only the information associated with symbolic state η_1 is detailed –note that values of clocks are actually summations of delays (e.g. $fClock_1$ represents the summation $0 + z_1$).

In order to deal with quiescence, we complete symbolic executions by new transitions. Contrarily to TIOLTS, transitions of TIOSTS carry actions that are necessarily inputs or outputs (not delays). For this reason, we artificially introduce a new symbol $\delta!$ denoting the absence of reactions.

Definition 12 (quiescence enrichment). *Let $SE(\mathbb{G}) = (Init, ST)$ be a symbolic execution. For all $\eta \in \mathcal{S}$ let us note $React(\eta)$ the set of st in ST with*

$$\begin{tikzcd}
 & & Init : (q_0, \theta_0, \pi_0, \vartheta_0, \lambda_0) \arrow[d, "z_0 \cdot location?loc_1"] & & & \\
 & & \eta_1 : (q_1, \theta_0, \pi_0, \vartheta_1, \lambda_1) \arrow[d, "z_1 \cdot plan!askFp_0"] & & & \\
 & & \eta_2 : (q_2, \theta_1, \pi_0, \vartheta_2, \lambda_2) \arrow[dl, "z_2 \cdot error!fTimeout_0"'] \arrow[dr, "z_3 \cdot plan?fPlan_1"] & & & \\
 & \eta_3 : (q_0, \theta_2, \pi_0, \vartheta_3, \lambda_3) \arrow[d, "z_4 \cdot location?loc_2"'] & & \eta_4 : (q_3, \theta_3, \pi_0, \vartheta_4, \lambda_4) \arrow[d, "z_5 \cdot param!askP_0"] & & \\
 & \eta_5 : (q_1, \theta_2, \pi_0, \vartheta_5, \lambda_5) \arrow[d] & & \eta_6 : (q_4, \theta_4, \pi_0, \vartheta_6, \lambda_6) \arrow[dl, "z_6 \cdot notif!cParams_0"'] \arrow[dr, "z_7 \cdot param?dParams_1"] & & \\
 & {} & \eta_7 : (q_5, \theta_5, \pi_0, \vartheta_7, \lambda_7) \arrow[d] & & \eta_8 : (q_5, \theta_6, \pi_0, \vartheta_8, \lambda_8) \arrow[d, "z_8 \cdot calc!data_1"] & \\
 & & {} & & \eta_9 : (q_6, \theta_7, \pi_0, \vartheta_9, \lambda_9) \arrow[dl, "z_9 \cdot error!cTimeout_0"'] \arrow[dr, "z_{10} \cdot calc?cmds_1"] & \\
 & & & \eta_{10} : (q_0, \theta_8, \pi_0, \vartheta_{10}, \lambda_{10}) \arrow[d] & & \eta_{11} : (q_7, \theta_9, \pi_0, \vartheta_{11}, \lambda_{11}) \arrow[d, "z_{11} \cdot nCmd!cmds_1"] \\
 & & & {} & & \eta_{12} : (q_0, \theta_{10}, \pi_0, \vartheta_{12}, \lambda_{12}) \arrow[d] \\
 & & & & & {}
\end{tikzcd}$$

$$\boxed{\begin{array}{rl}
\multicolumn{2}{l}{\eta_{12} : (q_0, \theta_{10}, \pi_0, \vartheta_{12}, \lambda_{12})} \\
\hline
\theta_{10} : & \top \wedge fClock_1 < 1 \wedge fClock_4 < 2 \wedge dClock_5 < 1 \\
 & \wedge dClock_7 < 1 \wedge cClock_8 < 1 \wedge cClock_9 < 2 \\
 & \wedge aClock_{11} < 9 \\
\pi_0 : & \top \\
\vartheta_{12} : & z_0 + z_1 + z_3 + z_5 + z_7 + z_8 + z_{10} + z_{11} \\
\lambda_{13} : & loc \leftarrow loc_1, askFp \leftarrow askFp_0, \\
 & fPlan \leftarrow fPlan_1, \ldots
\end{array}}$$

Fig. 3: Symbolic execution for the Trajectory module example.

$source(st) = \eta$ and $act(st) \in O(\Sigma_F)$. The quiescence enrichment *of $SE(\mathbb{G})$ is the couple $SE(\mathbb{G}_\delta) = (Init, ST \cup \Delta ST)$ where for all $\eta \in \mathcal{S}$:*
• **Time based quiescence:** *Let $\theta_\delta(\eta)$ be $\top$ if $React(\eta) = \emptyset$ and let $\theta_\delta(\eta)$ be $\bigwedge_{st \in React(\eta)} \forall delay(st).\neg(\theta(target(st)))$ otherwise. Then, $(\eta, \delta!, \eta_\delta^t) \in \Delta ST$ with $\eta_\delta^t = (q_\delta, \theta(\eta) \wedge \theta_\delta(\eta), \pi(\eta), \vartheta(\eta) + z, \lambda(\eta))$ with z is a new variable in F_{time}.*
• **Data based quiescence:** *Let $\pi_\delta(\eta)$ be $\top$ if $React(\eta) = \emptyset$ and let $\pi_\delta(\eta)$ be $\bigwedge_{st \in React(\eta)} \neg\pi(target(st))$ otherwise. Then, $(\eta, \delta!, \eta_\delta^d) \in \Delta ST$ with $\eta_\delta^d = (q_\delta, \theta(\eta), \pi(\eta) \wedge \pi_\delta(\eta), \vartheta(\eta) + z, \lambda(\eta))$ with z is a new variable in F_{time}.*

Time based quiescence transitions can be executed only if no transition labeled by an output can be executed anymore due to unsatisfiable time constraints. By noting that $delay(st)$ is the symbolic delay associated wit the transition st, $\theta_\delta(\eta)$ precisely states that for all output transitions st of source η, whatever the delay is, the time path condition $(\theta(target(st)))$ to fire st cannot be satisfied. Similarly, **Data based quiescence** transitions can be executed only if no transition labelled by an output can be executed anymore due to unsatisfiable data constraints.

Example 3 (Quiescence enrichment).
The *Trajectory* module example takes all possible deadlock situations into account. This is normal, since in an flight management system, we do not want to have any of those situations. For illustration purposes, let us consider $Init$, since there is no output transition leaving from it, a **Data based quiescence** is detected, and as according to Definition 12, the transition $(Init, \delta!, \eta_\delta^d)$ is added to the tree, representing the fact that the module can remain silent until it receives a location from the environment. If we consider η_1, let us examine if we can detect a **Time based quiescence**. That is, let us examine the formula

$\forall z_1.\neg(\theta_1)$, where $\theta_1 = fClock_1 < 1$. Since $fClock_1 = 0 + z_1$, there is no way that for all z_1, $z_1 \geq 1$ is true. Thus, no **Time based quiescence** is added to the tree.

$SE(\mathbb{G})_\delta$ characterizes in an intentional way the set of all timed traces of the TIOLTS associated to $\mathbb{G}$: we define paths of $SE(\mathbb{G})_\delta$ as finite sequences $st_1 \cdots st_n$ of transitions of ST, such that $source(st_1) = Init$ and for every $i < n$, we have $target(st_i) = source(st_{i+1})$. For any finite path $p = st_1 \cdots st_n$, $target(p)$ is $Init$ if $n = 0$ and $target(st_n)$ otherwise. We note $Path(SE(\mathbb{G})_\delta)$ the set of all such paths. For a path p, we note $Seq(p)$ the sequence defined as ε if $p = \varepsilon$, $Seq(p').delay(st)$ if p is of the form $p'.st$ with $act(st) = \delta!$ and $Seq(p').delay(st).act(st)$ if p is of the form $p'.st$ with $act(st) \neq \delta!$. The set $TTraces(p)$ is defined as $\{\sigma | \exists \nu.(M \models_\nu \theta(target(p)) \wedge \pi(target(p)) \wedge \overline{\sigma} = \nu(Seq(p)))\}$.[8] We note $TTraces(SE(\mathbb{G})_\delta)$ the set $\bigcup_{p \in Path(SE(\mathbb{G})_\delta)} TTraces(p)$.

5 Off-line Testing Algorithms

We present our algorithms w.r.t. the input sequence selection, the test execution, and the verdict computation.

Input sequence selection. We propose to extract input sequences, as introduced in Definition 4, from $SE(\mathbb{G})_\delta$. Following our previous works [7, 8], we propose to use paths of $SE(\mathbb{G})_\delta$ as *test purposes*, that is, we build input sequences corresponding to traces of a selected path. Given a test purpose p in $Path(SE(\mathbb{G})_\delta)$, an input sequence is built by applying the input projection on a trace chosen in $TTraces(p)$. Indeed, such input sequences are clearly good candidates to put the SUT $\mathbb{S}$ in a configuration where $\mathbb{S}$ can reach the test purpose. Therefore, we introduce the *set* $IS(p) = \{\sigma_{\downarrow_I} \mid \sigma \in TTraces(p)\}$ *of input sequences of* p. We can capture the set $IS(p)$ by simply building the corresponding normalized traces and by highlighting the interpretations which satisfy the constraints defining the target state of p:

$$\{\nu(Seq(p))_{\downarrow_I} \mid \nu \models \theta(target(p)) \wedge \pi(target(p))\}$$

Thus, defining a normalized input sequence for p comes to exhibit an interpretation $\nu : F \to M$ that satisfies both the time and data path conditions of the target of p by constraint solving techniques, to compute $\nu(Seq(p))$, and finally "forget" output in the result thanks to the $\downarrow_I$ operator. We suppose that an SUT $\mathbb{S}$, a test purpose p and an input sequence $\sigma^p_{\downarrow_I}$ are given (σ^p is thus a timed trace of p, normalized or not).

Test execution. As discussed in Section 3, the *test execution* is to submit $\sigma^p_{\downarrow_I}$ to $\mathbb{S}$, which in turn sends an output sequence σ_o, such that $\sigma^p_{\downarrow_I} \rightsquigarrow_{\mathbb{S}} \sigma_o$. We note $\sigma_{\mathbb{S}} = Merge(\sigma^p_{\downarrow_I}, \sigma_o)$. Recall that, due to potential non determinism of $\mathbb{G}$, even in the case $\mathbb{S}$ *tioco* $\mathbb{G}$, we may have $\sigma_{\mathbb{S}} \neq \sigma^p$.

[8] An interpretation ν can be extended to sequences as follows: $\nu(Seq(p)) = \varepsilon$ if $p = \varepsilon$, $\nu(Seq(p)) = \nu(\omega).\nu(a)$ if $Seq(p) = \omega.a$ with $a \in Act(\Sigma_F)$ and $\nu(Seq(p)) = \nu(\omega).\nu(d)$ if $Seq(p) = \omega.d$ with $d \in T_\Omega(F)_{time}$.

Verdict computation. Our algorithm takes as inputs three arguments, $SE(\mathbb{G})_\delta$, p and $\sigma_\mathbb{S}$, and computes verdicts concerning the correctness of $\mathbb{S}$ assessment and concerning the coverage of p by $\sigma_\mathbb{S}$. This algorithm can be seen as an off-line version of the one defined in [7]. We begin by introducing some intermediate definitions that are needed in the algorithm. A *context* is a mathematical structure denoting a path of $SE(\mathbb{G})_\delta = (Init, ST)$ potentially covered by a trace, together with additional identification constraints induced by the trace. Formally, a context is a tuple (η, f_t, f_d, d) where: $\eta \in \mathcal{S}$ denotes the target state of the potentially covered path; $f_t \in Sen_{Time}(F_{time})$ expresses identification constraints between time variables of F and numerical delays occurring in the trace; $f_d \in Sen_\Omega(F)$ expresses identification constraints between data variables and values emitted and received in the trace; and finally, d is a numerical delay that identifies how much time has elapsed in the given context. Intuitively, when time elapses, if there is no modification of state, then the value d is simply increased, else, the value d is reset to 0. In the sequel, $state((\eta, f_t, f_d, d))$ names the state η. Since there may be more than one path which is covered by a trace, we manipulate sets of contexts. We introduce the function $Next(a, SC)$, which computes the set of all contexts that can be reached from a given set of contexts SC, when an action occurs or a delay elapses, i.e. $a \in Act_M(C) \cup (D \setminus \{0\})$:
Case $a \in Act_M(C)$ of the form $c\triangle t$ with $\triangle \in \{?, !\}$: if there exists $(\eta, f_t, f_d, d) \in SC$, and a symbolic transition $st = (\eta, c\triangle u, \eta')$ in ST, then:

$$(\eta', f_t \wedge d = delay(st), f_d \wedge (t = u), 0) \in Next(a, SC)$$

provided that both $f_t \wedge d = delay(st) \wedge \theta(\eta')$ and $f_d \wedge \pi(\eta')$ are satisfiable.
Case $a \in D \setminus \{0\}$: if there exists $(\eta, f_t, f_d, d) \in SC$, and a symbolic transition st in ST with $source(st) = \eta$, then:
$(\eta, f_t \wedge \exists delay(st).(delay(st) \geq d + a \wedge \theta(target(st))), f_d, d + a) \in Next(a, SC)$
provided that $f_t \wedge \exists delay(st).(delay(st) \geq d + a \wedge \theta(target(st))) \wedge \theta(\eta)$ is satisfiable.

The general idea of the algorithm is to read one by one the elements of the trace $\sigma_\mathbb{S}$, and to compute either the next set of contexts or to emit a verdict. Let us suppose that $\sigma_\mathbb{S}$ can be written as $\sigma_{pref}.a.\sigma_{suf}$ where a is an action or a delay. $SC(a)$ is a notation grouping the set of contexts SC reached after reading the beginning $\sigma_{pref}.a$ of the trace and the last analyzed element a. At the initialization step, when no element of $\sigma_\mathbb{S}$ has been analyzed, then we use the symbol $\_$, that is $SC(\_)$. The algorithm is then given as a set of rules of the form:

$$\frac{SC(a) \quad \sigma_{suf}}{Result} \; cond$$

σ_{suf} is the remaining trace to be analyzed with respect to the first analyses stored in $SC(a)$, to $SE(\mathbb{G})_\delta$, and to p; *cond* is a set of conditions that has to be satisfied so that the rule can be applied; and *Result* is either a verdict or of the form $SC'(a') \;\; \sigma'_{suf}$. Moreover, if $\sigma_{suf} = \varepsilon$ then *Result* is necessarily a verdict since the initial trace $\sigma_\mathbb{S}$ is fully analyzed. If *Result* is $SC'(a') \;\; \sigma'_{suf}$, then σ_{suf} can be written as $a'.\sigma'_{suf}$. We will access respectively to a' and σ'_{suf} from σ_{suf} by using the usual notations $head(\sigma_{suf})$ and $tail(\sigma_{suf})$.

Rule 0 corresponds to the initialization phase: the set of contexts contains only one context stating that we begin at the symbolic state $Init$, there are no constraints identified yet, and the associated delay is 0. **Rule 1** is applied to compute a new set of contexts. This is done as long as SC is not empty and there are still elements of the trace to read. There are five verdicts: $FAIL$ is emitted when the trace denotes an incorrect behavior (**Rule 2**); $PASS$ is emitted when the trace denotes a correct behaviors, and the test purpose is the only path covered (**Rule 3**); $WEAK\_PASS$ is emitted when the trace denotes a correct behavior, the test purpose is covered but there exists at least one other path covered (**Rule 4**); $INCONC_r$ is emitted when the trace denotes a correct behavior, a path is covered but not the test purpose (**Rule 5**); $INCONC_i$ is emitted when the trace is not included in $SE(\mathbb{G})_\delta$ due to input under-specification (**Rule 6**).

Rule 0: Initialization

$$\frac{}{\{(Init, \top, \top, 0)\}(\_) \quad \sigma_{\mathbb{S}}}$$

Rule 1: An action or a delay is read from the trace, SC is not empty.

$$\frac{SC(a) \quad \sigma}{Next(head(\sigma), SC)(head(\sigma)) \quad tail(\sigma)} \; SC \neq \emptyset, \;\; \sigma \neq \epsilon$$

Rule 2: An unspecified output o or delay d is read from the trace.

$$\frac{SC(a) \quad \sigma}{FAIL} \; SC = \emptyset; \;\; a \in O_M(C) \cup D \setminus \{0\}$$

Rule 3: The read action permits to cover the test purpose, and no other paths.

$$\frac{SC(a) \quad \sigma}{PASS} \; \sigma = \epsilon; \forall ct \in SC, state(ct) = target(p); SC \neq \emptyset$$

Rule 4: The read action permits to cover the test purpose, and at least one other path.

$$\frac{SC(a) \quad \sigma}{Weak\_PASS} \; \begin{array}{c} \sigma = \epsilon; \exists ct \in SC, state(ct) = target(p); \\ \exists ct' \in SC, state(ct') \neq target(p); SC \neq \emptyset \end{array}$$

Rule 5: Some paths are covered but not the test purpose.

$$\frac{SC(a) \quad \sigma}{INCONC_r} \; \sigma = \epsilon; \forall ct \in SC, state(ct) \neq target(p); SC \neq \emptyset$$

Rule 6: An unspecified input i is read.

$$\frac{SC(a) \quad \sigma}{INCONC_i} \; SC = \emptyset; \;\; a \in I_M(C)$$

In contrast with algorithms in [7,8], there are two kinds of inconclusive verdicts. $INCONC_r$ corresponds to the classical one, while $INCONC_i$ is produced when $\sigma_{\mathbb{S}}$ is of the form $\sigma_{pref}.a.\sigma_{suf}$, where σ_{pref} is in $SE(\mathbb{G})_\delta$ while $\sigma_{pref}.a$ with $a \in I_M(C)$ is not. This situation does not occur in on-line testing algorithms, because they are in charge of stimulating the SUT so that it strictly follows the test purpose, and emits a verdict as soon as the test purpose cannot be covered anymore. On the contrary, unless $\sigma_{\mathbb{S}}$ can be decomposed as $\sigma_{pref}.a.\sigma_{suf}$, where σ_{pref} is a specified timed trace and $\sigma_{pref}.a$ is not (in which case either $FAIL$ or $INCONC_i$ depending on the nature of a), all actions of $\sigma_{\mathbb{S}}$ will be analyzed even though the emission of $PASS$ is not possible anymore. This choice is made in order to always emit $FAIL$ for a trace revealing a non conformance, even if we may be sure (several steps before) that $PASS$ can not be emitted anymore.

Example 4. Let us apply our rule-based algorithm to the *Trajectory* module example. Let us suppose that there is a mapping between messages and integers. Then, we choose the path of Figure 3 leading to η_{13}, representing a complete loop for the module, i.e., the path p:
$z_0 \cdot location?loc_1 \cdot z_1 \cdot plan!askFp_0 \cdot z_3 \ldots z_8 \cdot calc!data_1 \cdot z_{10} \cdot calc?cmds_1 \cdot z_{12} \cdot nCmd!cmds_1$.
Thus, the tester chooses the appropriate values and performs the *input projection* of the trace, obtaining $\sigma^p_{\downarrow I}$: $0.1\ location?4\ 2.8\ plan?5\ 1.8\ param?2\ 1\ calc?1$. Let us assume that the SUT *responds* with $\sigma_{\downarrow O}$: $0.2\ plan!8\ 2.1\ error!6\ 0.1\ nCmd!3\ 4.3$. By applying the merge operation on $\sigma^p_{\downarrow I}$ and $\sigma_{\downarrow O}$, we obtain the trace σ:
$0.1\ location?4\ 0.1\ plan!8\ 2.1\ error!6\ 0.1\ nCmd!2\ 0.5\ plan?5\ 2\ param?2\ 1\ calc?1\ 1$.
Figure 4 illustrates the application of the rule-based algorithm to σ.

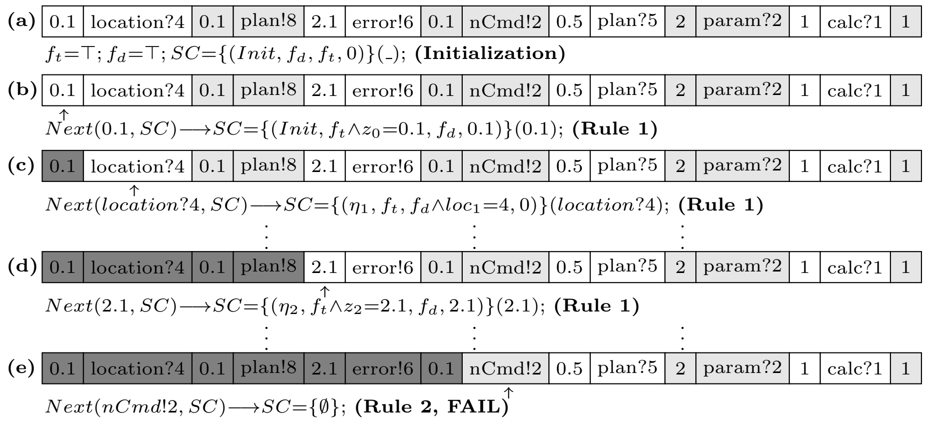
(a) 0.1 location?4 0.1 plan!8 2.1 error!6 0.1 nCmd!2 0.5 plan?5 2 param?2 1 calc?1 1
$f_t=\top$; $f_d=\top$; $SC=\{(Init, f_d, f_t, 0)\}(\_)$; (Initialization)
(b) 0.1 location?4 0.1 plan!8 2.1 error!6 0.1 nCmd!2 0.5 plan?5 2 param?2 1 calc?1 1
$Next(0.1, SC) \longrightarrow SC=\{(Init, f_t \wedge z_0=0.1, f_d, 0.1)\}(0.1)$; (Rule 1)
(c) 0.1 location?4 0.1 plan!8 2.1 error!6 0.1 nCmd!2 0.5 plan?5 2 param?2 1 calc?1 1
$Next(location?4, SC) \longrightarrow SC=\{(\eta_1, f_t, f_d \wedge loc_1=4, 0)\}(location?4)$; (Rule 1)
(d) 0.1 location?4 0.1 plan!8 2.1 error!6 0.1 nCmd!2 0.5 plan?5 2 param?2 1 calc?1 1
$Next(2.1, SC) \longrightarrow SC=\{(\eta_2, f_t \wedge z_2=2.1, f_d, 2.1)\}(2.1)$; (Rule 1)
(e) 0.1 location?4 0.1 plan!8 2.1 error!6 0.1 nCmd!2 0.5 plan?5 2 param?2 1 calc?1 1
$Next(nCmd!2, SC) \longrightarrow SC=\{\emptyset\}$; (Rule 2, FAIL)

Fig. 4: Off-line Test Algorithm operating on a trace of the *Trajectory* module. At any iteration, before the execution of any rule, σ is the non-dark-gray-shadowed part of the depicted trace, $head(\sigma)$ is the element pointed by the arrow, a is the first element at its left (when it applies), and $queue(\sigma)$ is the right-side remainder. In **(a)**, SC represents the set of context after initialization. **(b)**–**(d)** represent the application of **Rule 1**, updating SC by reading the first element of the trace (and applying the $Next()$ function). In **(e)**, the observed delay causes the verdict to emit $FAIL$, since $Next()$ returns an empty set.

What is interesting to notice in Figure 4 is that, if use an on-line test purpose-guided algorithm as in [7], we emit the verdict $INCONC$ as soon as we find the delay 2.1 of output $error!6$ (**(d)** in the figure). One advantage of this off-line algorithm is that we can emit better verdicts (less $INCONC$) since we are analyzing the entire trace.

6 Related Work

Model-based conformance testing with *tioco* was addressed in [4, 5, 7] where models are essentially timed automata (TA) [1] (without symbolic data). Authors in [5] have defined a pure on-the-fly testing algorithm. Without any preprocessing on the model, at each moment, the algorithm computes on-the-fly a random input and its corresponding submission delay from the model and checks outputs and their timing against the model. This is reiterated until a verdict is emitted. In [4], authors model the testing activity with the help of test cases which are deterministic timed automata with inputs and outputs, whose states are labeled by verdicts. These test cases are derived from a given test purpose and result from some approximate determinisation mechanisms preserving the *tioco* conformance relation. Such a design of test cases tends to reduce choice points at runtime: however, decisions have still to be made when running test cases on SUT, for example, concerning the choice between waiting for a delay or sending a data to the SUT. So [4] may be viewed as a mixed approach, off-line for the selection, by converting a test purpose as a test case, on-line for guiding the progress in the test case during the execution. Recently, authors of [2,3,7,15] have defined seemingly different reference models where both time and data are represented symbolically as extensions to TA or/and input output symbolic transition systems (IOSTS) [8,13], still based on *tioco*. In addition, approaches [2,7] have suggested on-line testing algorithms guided by a test purpose: in [7], symbolic execution paths are selected as test purposes while the work in [2] is conducted in the spirit of [4]. To our knowledge, we are the first to propose in the context of the *tioco* conformance relation a framework where test inputs are presented statically, without any further processing to be done, while remaining executable and usable for verdict computation.

7 Conclusion

We have proposed an off-line testing approach based on the *tioco* conformance relation and on TIOSTS models which handle both data and time symbolically. The approach includes three steps: (1) first, test input sequences are extracted from a TIOSTS; for this, traces are extracted from paths of the symbolic execution tree by using solving constraints and projection techniques; (2) test executions produce output sequences that are merged with input sequences to form input output traces; (3) resulting traces are analyzed in order to provide verdicts. We highlight a verdict, specific to our off-line approach: the verdict $INCONC_i$, stating that an input can become unspecified in the context of the test execution even if it was a specified input in the context of the test selection. Our approach has been implemented using the symbolic execution tool Diversity. Diversity is an extension of the tool AGATHA [12] that integrates several sat-solvers and can analyze several languages, in particular TIOSTS or UML sequence diagrams extended with time constraints [3,7].[9] Concerning data, our implementation handles booleans, presburger integers, and could be extended to any decidable data theory as in [8]. Until now, we have not investigaged zone-based techniques usually undertaken with timed automata, essentially because run-time efficiency is not of primary importance in an off-line framework. We

[9] *e.g.* CVC3 $http://www.cs.nyu.edu/acsys/cvc3/$

are investigating techniques to reduce the occurrence of inconclusive verdicts. As several reactions are possible after a given trace, the submission of an input sequence to SUT may result in a trace which runs outside the test purpose without questioning conformance. To control non-determinism as much as possible, we are studying under which hypotheses we can over constrain the input sequence computation in order to force a correct SUT to follow a given path.

References

1. R. Alur and D.L. Dill. The Theory of Timed Automata. In *Proc. of REX Workshop the Real-Time: Theory in Practice*. Springer, 1992.
2. W.L. Andrade, P.D.L. Machado, T. Jéron, and H. Marchand. Abstracting Time and Data for Conformance Testing of Real-Time Systems. In *Proc. of Int. Conf. Software Testing, Verification and Validation (ICSTW) Workshops*. IEEE, 2011.
3. B. Bannour, C. Gaston, and D. Servat. Eliciting unitary constraints from timed Sequence Diagram with symbolic techniques: application to testing. In *Proc. of Int. Conf. Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2011.
4. N. Bertrand, T. Jeron, A. Stainer, and M. Krichen. Off-line test selection with test purposes for non-deterministic timed automata. In *Proc. of Int. Conf. Tools and algorithms for the construction and analysis of systems/European conferences on theory and practice of software (TACAS/ETAPS)*. Springer, 2011.
5. H.C. Bohnenkamp and A. Belinfante. Timed Testing with TorX. In *Proc. of Int. Conf. Formal Methods Europe (FM)*. Springer, 2005.
6. L.B. Briones and E. Brinksma. A Test Generation Framework for *quiescent* Real-Time Systems. In *Proc. of Int. Conf. Formal Approaches to Software Testing (FATES)*. Springer, 2004.
7. J.P. Escobedo, C. Gaston, and P. Le Gall. Timed Conformance Testing for Orchestrated Service Discovery. In *Proc. of Int. Conf. Formal Aspects of. Component Software (FACS)*. Springer, 2011.
8. C. Gaston, P. Le Gall, N. Rapin, and A. Touil. Symbolic Execution Techniques for Test Purpose Definition. In *Proc. of Int. Conf. Testing of Software and Communicating Systems (TestCom)*, pages 1–18. Springer, 2006.
9. M. Krichen and S. Tripakis. Black-box time systems. In *Proc. of Int. SPIN Workshop Model Checking of Software*. Springer, 2004.
10. B. Marre and A. Arnould. Test Sequences Generation from LUSTRE Descriptions: GATeL. In *Proc. of Int. Conf. Automated Software Engineering (ASE)*. IEEE, 2000.
11. A. Petrenko and N. Yevtushenko. Testing from Partial Deterministic FSM Specifications. *IEEE Trans. Comput.*, 2005.
12. N. Rapin, C. Gaston, A. Lapitre, and J.-P. Gallois. Behavioural unfolding of formal specifications based on communicating automata. In *Proc. of Workshop on Automated technology for verification and analysis (ATVA)*, 2003.
13. V. Rusu, L. Bousquet, and T. Jéron. An Approach to Symbolic Test Generation. In *Proc. of Int. Conf. Integrated Formal Methods (IFM)*. Springer, 2000.
14. J. Schmaltz and J. Tretmans. On Conformance Testing for Timed Systems. In *Proc. of Int. Conf. Formal Modeling and Analysis of Timed Systems (FORMATS)*. Springer, 2008.
15. S. Von Styp, H. Bohnenkamp, and J. Schmaltz. A conformance testing relation for symbolic timed automata. In *Proc. of Int. Conf. Formal modeling and analysis of timed systems (FORMATS)*. Springer, 2010.