Non-Deterministic Timed Pushdown Automata-Based Testing Evaluated by Mutation

Hana M'Hemdi*†, Jacques Julliand*, Pierre-Alain Masson* and Riadh Robbana‡
* FEMTO-ST/DISC, University of Franche-Comté
16, route de Gray F-25030 Besançon Cedex France
{hana.mhemdi, jacques.julliand, pierre-alain.masson}@femto-st.fr
† LIP2, University of Tunis El Manar, Tunisia
‡ LIP2 and INSAT-University of Carthage, Tunisia
riadh.robbana@fst.rnu.tn

***Abstract*—This paper is about conformance testing of non deterministic timed pushdown automata with inputs and outputs (TPAIO), that specify both stack and clock constraints. It proposes a novel off-line test generation method from this model. The first step computes a deterministic timed pushdown tester with inputs and outputs (DTPTIO): a TPAIO which approximates the initial TPAIO with only one clock. Then we compute from the DTPTIO a finite reachability automaton (RA), whose transitions are related to DTPTIO paths satisfying the stack constraints. This computation takes the DTPTIO transitions as a coverage criterion. The RA transitions, thus the DTPTIO paths, are used for generating test cases that aim at covering the reachable locations and transitions of the TPAIO. The test cases are in the shape of trees equipped with verdicts. Last, we propose a mutation testing method from non-deterministic TPAIO to evaluate the efficiency of our method.**

Keywords–Timed Automata; Timed Pushdown Automata; Conformance testing; Reachability automata; Model-based Mutation.

I. INTRODUCTION

Systems are commonly modelled by means of transition systems such as finite automata, timed automata, etc. System formal verification as well as model-based test generation, that are very active research areas, rely on exploiting these models. This paper is about generating tests from the model of non deterministic timed pushdown automata [1] with inputs and outputs (*TPAIO*). Timed automata (*TA*) are equipped with a finite set of real-valued clocks. They have been introduced by Alur and Dill [2], and have become a standard for modelling real-time systems. Pushdown automata (*PA*) [3] are equipped with a stack, and can be used for modelling recursive systems. The model of *TPAIO* includes both clocks and a stack.

Test generation from *TPAIO* could apply to industrial case studies such as that of [4], that defines automatic synthesis of robust and optimal controllers. These kinds of controllers operate on variables that are constantly growing in real-time, such as oil pressure etc. As shown in [5], this system can be modelled as a recursive timed automaton with a safety critical objective. [6] argues that timed recursive state machines are related to an extension of timed pushdown automata, where an additional stack, coupled with the standard control stack, is used to store temporal valuations of clocks. Therefore, the system of [4] can be modelled by means of a *TPAIO*. Also, *TPAIO* can serve as a model for the verification of real-time distributed systems, as quoted in [6].

We propose in this paper an approach for computing offline tests from a *TPAIO* model. Offline tests, contrarily to online tests that are dynamically computed along an execution, are first extracted out of the model as a set of abstract executions, to be subsequently executed on the system. We focus in this paper on testing non deterministic timed pushdown automata with inputs and outputs. We propose a new approach for conformance testing of *TPAIO*, aiming at covering its reachable locations and transitions. The idea is to deal successively with the clock and stack constraints, that could prevent a location from being reachable. Location reachability in *TPAIO* is decidable [7][1], and time exponential [8].

Our first contribution is to reuse and adapt a determinization method [9] for *TAIO* (timed IO automata with no stack) in the *TPAIO* case. We obtain a single clock deterministic timed pushdown tester with inputs and outputs (*DTPTIO*), in which the locations reachability has been verified w.r.t. the satisfiability of the clock constraints. Fail verdicts are added as special locations to this *DTPTIO*: they model the observation of timeouts or of unspecified stack and output actions.

Locations reachability has further to be verified w.r.t. the stack constraints. Finkel et al. propose in [10] a polynomial method for checking locations reachability in a *PA*. It relies on a set of rules that compute, in the shape of a finite automaton, a reachability automaton (*RA*) from a *PA*. As a second contribution we propose, following them, to adapt the rules to the *TPAIO* case, with a transition coverage criterion. We compute an *RA* where each transition is related to one path of the *DTPTIO*. The paths are used to extract a set of tests out of the original *TPAIO*: we expand the paths that reach a final location from an initial one with an empty stack. This computes a tree of test cases from a *TPAIO*.

Model-based mutation testing [11] is a method to evaluate the efficiency of model-based test suites. The specification model is intentionally mutated to generate altered models called mutants, either correct or incorrect. Mutation-based testing techniques aim at measuring how well the test cases generated from the specification detect non-conformance between the specification and its mutants. Our third contribution is the definition of *TPAIO* mutation operators. We apply them to a *TPAIO* example, and evaluate the success of our non-conformance detection technique.

To summarize, our contributions are to: (i) define $tpioco$, a conformance relation for the *TPAIO* model; (ii) adapt the

determinization method of [9] to obtain a *DTPTIO*; (iii) compute an *RA* whose transitions are each labelled with a a path of a *DTPTIO*, by adapting the reachability computation of [10]; (iv) generate a tree of test cases by covering the reachable locations and transitions of the *TPAIO*: to our knowledge these problems, solved for TA and PA, have not yet been handled for *TPAIO*; (v) evaluate the efficiency of our method by mutation techniques.

The paper is organized as follows. Section II presents the *TA* model and the timed input-output conformance relation $tioco$. Our *TPAIO* model is presented in Section III, together with a *TPAIO* conformance relation and *TPAIO* mutation operators. Our test generation method is presented in Section IV. We discuss its soundness and coverage in Section V. The tests execution principle on an implementation as well as experimental results on a *TPAIO* example are given in Section VI. Section VII concludes the paper and indicates future research directions.

II. Background

This section defines *TA* and a timed input-output conformance relation.

A. Timed Automata

Let $Grd(X)$ be the language of clock guards defined as conjunctions of expressions $x \sharp n$ where x is a clock of X, n is a natural integer constant and $\sharp \in \{<, \leq, >, \geq, =\}$. Let $CC(X)$ be the language of clock constraints defined as conjunctions of expressions $e \sharp n$ where e is either x, $x - x'$ or $x + x'$.

Definition 1 (Timed Automaton): A *TA* is a tuple $T = \langle L, l_0, \Sigma, X, \Delta, F\rangle$ where L is a finite set of locations, l_0 is an initial location, Σ is a finite set of labels, X is a finite set of clocks, $F \subseteq L$ is a set of accepting locations and $\Delta \subseteq L \times \Sigma \times Grd(X) \times 2^X \times L$ is a finite set of transitions.

A transition is a tuple (l, a, g, X', l') denoted by $l \xrightarrow{a,g,X'} l'$ where $l, l' \in L$ are respectively the source and target locations, a ($\in \Sigma$) is an action symbol, X' ($\subseteq X$) is a set of resetting clocks and g is a guard. The operational semantics of a *TA* T is an infinite transition system $\langle S^T, s_0^T, \Delta^T\rangle$ where the states of S^T are pairs $(l, v) \in L \times (X \to \mathbb{R}^+)$, with l a location and v a clock valuation. s_0^T is the initial state and Δ^T is the set of transitions. There are two kinds of transitions in Δ^T: timed and discrete. Timed transitions are in the shape of $(l, v) \to^\delta (l, v + \delta)$ where $\delta \in \mathbb{R}^+$ is a delay, so that $v + \delta$ is the valuation v with each clock augmented by δ. Discrete transitions are in the shape of $(l, v) \to^a (l', v')$ where $a \in \Sigma$ and $(l, a, g, X', l') \in \Delta$, and such that v satisfies g and $v' = v[X' := 0]$ is obtained by resetting to zero all the clocks in X' and leaving the others unchanged. A path π of a *TA* is a finite sequence of its transitions: $l_0 \xrightarrow{a_0,g_0,X_0} l_1 \xrightarrow{a_1,g_1,X_1} l_2 \cdots l_{n-1} \xrightarrow{a_{n-1},g_{n-1},X_{n-1}} l_n$. A run of a *TA* is a path of its semantics. It alternates timed and discrete transitions. $\sigma = (l_0, v_0) \to^{\delta_0} (l_0, v_0 + \delta_0) \to^{a_0} (l_1, v_1) \to^{\delta_1} (l_1, v_1 + \delta_1) \to^{a_1} (l_2, v_2) \to^{\delta_2} \ldots \to^{a_{n-1}} (l_n, v_n)$ where $\delta_i \in \mathbb{R}^+$ and $a_i \in \Sigma$ for each $0 \leqslant i \leqslant n-1$ is a run of π if v_i satisfies g_i for $0 \leqslant i < n$. Its trace is a finite sequence $\rho = \delta_0 a_0 \delta_1 a_1 \ldots \delta_n a_n$ of $(\Sigma \cup \mathbb{R}^+)^*$. We denote $RT(\Sigma)$ the set of finite traces $(\Sigma \cup \mathbb{R})^*$ on Σ. $P_{\Sigma_1}(\rho)$ is the projection on $\Sigma_1 \subseteq \Sigma$ of a trace ρ with the delays preserved. For example, if $\rho = 5a4b2$, then, $P_{\{a\}}(\rho) = 5a42$ i.e. $5a6$. $Time(\rho)$ is the sum of all the delays of ρ. For example, $Time(5a42) = 11$. $s_0^T \to^\rho s$ means that the state s is reachable from the initial state s_0^T, i.e. there exists a run σ from s_0^T to s whose trace is ρ. $s_0^T \to^\rho$ means that there exists s' such that $s_0^T \to^\rho s'$.

Timed Automata with Inputs and Outputs (*TAIO*) extend the *TA* model by distinguishing between input and output actions. A *TAIO* is a tuple $\langle L, l_0, \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}, X, \Delta, F\rangle$ where Σ_{in} is a set of input actions, Σ_{out} is a set of output actions and τ is an internal and unobservable action. This model is widely used in the domain of test. It models the controllable ($\in \Sigma_{in}$) and observable ($\in \Sigma_{out}$) interactions between the environment and the system. The environment, thus the tester, sends commands of Σ_{in} and observes outputs of Σ_{out}. The implementation under test (IUT), sends observable actions of Σ_{out} and accepts commands of Σ_{in}.

Let $\Sigma = \Sigma_{in} \cup \Sigma_{out}$ and $\Sigma_\tau = \Sigma \cup \{\tau\}$. A *TAIO* is deterministic if for all locations l in L, for all actions a in Σ_τ and for all couples of distinct transitions $t_1 = (l, a, g_1, X_1, l_1)$ and $t_2 = (l, a, g_2, X_2, l_2)$ in Δ then $g_1 \wedge g_2$ is not satisfiable. It is observable if no transition is labelled by τ. $Reach(T) = \{s^T \in S^T \mid \exists \rho.(\rho \in RT(\Sigma) \wedge s_0^T \to^\rho s^T\}$ denotes the set of reachable states of a *TAIO* T. A *TAIO* T is non blocking if $\forall (s, \delta).(s \in Reach(T) \wedge \delta \in \mathbb{R}^+ \Rightarrow \exists \rho.(\rho \in RT(\Sigma_{out} \cup \{\tau\}) \wedge Time(\rho) = \delta \wedge s \to^\rho))$. A *TAIO* is called input-complete if it accepts any input at any state.

B. Timed Input-Output Conformance Relation $tioco$

We first present the conformance theory for timed automata based on the conformance relation *tioco* [9]. $tioco$ is an extension of the *ioco* relation of Tretmans [12]. The main difference is that $ioco$ uses the notion of quiescence, contrarily to $tioco$ in [9] where the timeouts are explicitly specified. The assumptions are that the specification of the IUT is a non-blocking *TAIO*, and the implementation is a non-blocking and input-complete *TAIO*. This last requirement ensures that the execution of a test case on the IUT does not block the verdicts to be emitted.

To present the conformance relation for a *TAIO* $T = \langle L, l_0, \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}, X, \Delta, F\rangle$, we need to define the following notations in which $\rho \in RT(\Sigma_{in} \cup \Sigma_{out})$:

- T *after* ρ = $\{s \in S^T \mid \exists \rho'.(\rho' \in RT(\Sigma_\tau) \wedge s_0^T \to^{\rho'} s \wedge P_\Sigma(\rho') = \rho)\}$ is the set of states of T that can be reached by a trace ρ' whose projection $P_\Sigma(\rho')$ on the controllable and observable actions is ρ.
- *ObsTTraces*(T) = $\{P_\Sigma(\rho) \mid \rho \in RT(\Sigma_\tau) \wedge s_0^T \to^\rho\}$ is the set of observable timed traces of a *TAIO* T.
- *elapse*(s) = $\{\delta \mid \delta > 0 \wedge \exists \rho.(\rho \in RT(\{\tau\}) \wedge Time(\rho) = \delta \wedge s \to^\rho)\}$ is the set of delays that can elapse from the state s with no observable action.
- *out*(s) = $\{a \in \Sigma_{out} \mid s \to^a\} \cup elapse(s)$ is the set of outputs and delays that can be observed from the state s.

Definition 2 (tioco): Let $T = (L, l_0, \Sigma_\tau, X, \Delta, F)$ be a specification and $I = (L^I, l_0^I, \Sigma_\tau^I, X^I, \Delta^I, F^I)$ be an implementation of T. Formally, I conforms to T, denoted I *tioco* T iff $\forall \rho.(\rho \in ObsTTraces(T) \implies out(I \textit{ after } \rho) \subseteq out(T \textit{ after } \rho))$.

It means that the implementation I conforms to the specification T if and only if after any timed trace enabled in T, each output or delay of I is specified in T.

III. *TPAIO* Model and Conformance Testing

We define in this section *TPAIO*, our model, as well as *tpioco*, a conformance relation for *TPAIO*. We also propose mutation operators for *TPAIO*.

A. Model and Conformance Testing

A *TPAIO* is a *TAIO* equipped with a stack, whose alphabet is denoted by Γ. Its operational semantics is a transition system $\langle S^T, s_0^T, \Delta^T \rangle$ whose locations –called states– are configurations made of three components (l, v, p) with l a location of the *TPAIO*, v a clock valuation in $X \rightarrow \mathbb{R}^+$ and p a stack content in Γ^*.

Definition 3 (TPAIO): A *TPAIO* is a tuple $\langle L, l_0, \Sigma, \Gamma, X, \Delta, F \rangle$ where L is a finite set of locations, l_0 is an initial location, $\Sigma = \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}$ where Σ_{in} is a finite set of input actions, Σ_{out} is a finite set of output actions and $\{\tau\}$ is an internal and unobservable action, Γ is a stack alphabet ($\Sigma_{out} \cap \Sigma_{in} = \varnothing$ and $\Sigma \cap \Gamma = \varnothing$), X is a finite set of clocks, $F \subseteq L$ is a set of accepting locations, $\Delta \subseteq L \times (\Sigma_{in} \cup \Sigma_{out} \cup \Gamma^{+-}) \times Grd(X) \times 2^X \times L$ is a finite set of transitions where $\Gamma^{+-} = \{a^+ \mid a \in \Gamma\} \cup \{a^- \mid a \in \Gamma\}$. The symbols of Γ^{+-} represent either a push operation (of the symbol a) denoted a^+, or a pop operation denoted a^-. A transition is a tuple (l, a, g, X', l') denoted by $l \xrightarrow{a,g,X'} l'$ where $l, l' \in L$ are respectively the source and target locations, $a \in \Sigma \cup \Gamma^{+-}$ is either a label or a stack action, X' ($\subseteq X$) is a set of resetting clocks and g is a guard. There are two kinds of transitions in the semantics, timed and discrete. Timed transitions are in the shape of $(l, v, p) \rightarrow^{\delta} (l, v + \delta, p)$. For a transition (l, act, g, X', l'), there are three types of discrete transitions when v satisfies g: (1) *push* when $act = a^+$: $(l, v, p) \rightarrow^{a^+} (l', v[X' := 0], p.a)$ where $a \in \Gamma$, (2) *pop* when $act = a^-$: $(l, v, p.a) \rightarrow^{a^-} (l', v[X' := 0], p)$ where $a \in \Gamma$, (3) *output, input or internal* when $act = A \in \Sigma$: $(l, v, p) \rightarrow^{A} (l', v[X' := 0], p)$. A *TPAIO* is normalized if it executes separately push, pop and Σ operations. Any *TPAIO* can be normalized since any *PA* can be normalized [13]. In the remainder of the paper, we use a to denote the actions of Γ and A to denote the actions of $\Sigma_{out} \cup \Sigma_{in}$.

Example 1 (Example of a non-Deterministic TPAIO): Figure 1 shows a non-deterministic *TPAIO* where $\Sigma_{in} = \{A, D, F, G\}$, $\Sigma_{out} = \{B, C, E\}$ and $\Gamma = \{pow\}$. We use the notation $!A$ to denote A as an output action of Σ_{out} and $?A$ to denote A as an input action.

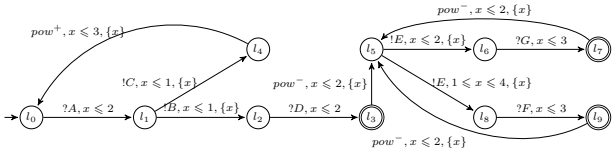

Figure 1. Example of a non-Deterministic *TPAIO*

As usual in conformance testing, we consider the IUT, the specification and the test cases to be all modelled by means of *TPAIO*.

Definition 4 (Parallel Composition of TPAIO): Let $I = \langle L^I, l_0^I, \Sigma_{in}^I \cup \Sigma_{out}^I \cup \{\tau\}, \Gamma, X^I, \Delta^I, F^I \rangle$ be an implementation and $TC = \langle L^T, l_0^T, \Sigma_{in}^T \cup \Sigma_{out}^T, \Gamma, X^T, \Delta^T, F^T \rangle$ be a test case such that $\Sigma_{in}^I = \Sigma_{out}^T$ and $\Sigma_{out}^I = \Sigma_{in}^T$ and $X^I \cap X^T = \varnothing$. The parallel composition $I \parallel TC$ is a *TPAIO* $\langle L^I \times L^T, (l_0^I, l_0^T), \Sigma_{in}^I \cup \Sigma_{out}^I \cup \{\tau\}, \Gamma, X^I \cup X^T, \Delta, F^I \times F^T \rangle$ where Δ is the smallest relation such that:

- $((l^I, l^T), act, g^I \wedge g^T, X'^I \cup X'^T, (l'^I, l'^T)) \in \Delta$ if $(l^I, act, g^I, X'^I, l'^I) \in \Delta^I$ and $(l^T, act, g^T, X'^T, l'^T) \in \Delta^T$ and $act \in \Sigma_{in}^I \cup \Sigma_{out}^I$ or $act \in \Gamma^{+-}$,
- $((l^I, l^T), \tau, g^I, X'^I, (l'^I, l^T)) \in \Delta$ if $(l^I, \tau, g^I, X'^I, l'^I) \in \Delta^I$

We define our conformance relation for *TPAIO*, denoted *tpioco*, as a particular case of the conformance relation *tioco* [9]. It is the same relation with the whole alphabet being $\Sigma_{in} \cup \Sigma_{out} \cup \Gamma^{+-} \cup \{\tau\}$ instead of just $\Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}$, and the output alphabet being $\Sigma_{out} \cup \Gamma^{+-}$ instead of just Σ_{out}.

B. TPAIO Mutation Operators

Given a *TPAIO* $T = \langle L, l_0, \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}, \Gamma, X, \Delta, F \rangle$, its mutants are defined by applying one of the following mutation operators:

- **Change action**: this operator changes only one transition in Δ by replacing its label by another one, either of an output or stack action. The obtained mutant $M = (L, l_0, \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}, \Gamma, \Delta \setminus \{t\} \cup \{t_M\}, F)$, such that $t = (l, a, g, X', l') \in \Delta$, $t_M = (l, b, g, X', l')$, $b \in \Sigma_{out} \cup \Gamma^{+-}$ and $a \neq b$.
- **Change guard**: this operator changes only one transition in T by replacing one relational operator in its guard. The obtained mutant $M = (L, l_0, \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}, \Gamma, \Delta \setminus \{t\} \cup \{t_M\}, F)$, such that $t = (l, a, g, X', l') \in \Delta$, $t_M = (l, a, g_m, X', l')$ where $a \in \Sigma_{out} \cup \Gamma^{+-}$, $g = \bigwedge_i x_i \sharp_i n_i \wedge x \sharp n$, $g_m = \bigwedge_i x_i \sharp_i n_i \wedge x \sharp_m n$, $\sharp, \sharp_i, \sharp_m \in \{<, \leq, >, \geq, =\}$ and $\sharp_m \neq \sharp$.
- **Negate Guard**: this operator changes only one transition in T by replacing its guard by its negation. The obtained mutant $M = (L, l_0, \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}, \Gamma, \Delta \setminus \{t\} \cup \{t_M\}, F)$, such that $t = (l, a, g, X', l') \in \Delta$, $t_M = (l, a, \neg g, X', l')$ and $a \in \Sigma_{out} \cup \Gamma^{+-}$.

IV. Test Generation from *TPAIO*

This section presents our test generation method from non-deterministic *TPAIO*. We first present the test generation process and then the three steps of our method.

A. Test Generation Process

The three steps of the test generation process that we propose in this paper are the following:

1) *TPAIO* determinization: off-line tests can be generated only from deterministic or determinizable systems. For this reason, we determinize a given *TPAIO* to obtain a Deterministic Timed Pushdown Tester with Inputs and Outputs (*DTPTIO*). The tester obtained is a deterministic *TPAIO* with one clock, reset each time the tester observes an action, and provided with $fail$ locations.

2) *RA* computation from the *DTPTIO*: this step computes one or many paths between two symbolic locations of a *DTPTIO* that satisfy the stack constraints, i. e. such that the stack content is the same in both the source and target locations. The *RA* is a finite automaton that maps any transition of the *RA* –called a π-transition– to a path π of the *DTPTIO*.
3) tree of test cases generation as nominal behaviours of the *TPAIO*, computed from the *DTPTIO*. There are two sub-steps: (a) generation of a tree of test paths by unfolding the paths π of the π-transitions that go from an initial to a final location of the *RA*; (b) generation of a tree of test cases of the *DTPTIO*, by adding to the test paths the $fail$ locations and the transitions that lead to it.

B. Construction of a DTPTIO from a TPAIO

We use the one-clock determinization of Krichen and Tripakis [9] to build a *DTPTIO* from a *TPAIO*. It relies on computing equivalence classes of clock values ensuring the same guards enabledness, in order to capture them with only one clock. Locations are symbolic (i.e. extended with clock constraints) and *fail* locations are added to model the occurrence of unspecified observations. The method of [9] is for *TAIO* (without stacks) and we have adapted it to the *TPAIO* case. Due to lack of space, we cannot go into the technical details, and only summarize our adaptation as follows:

- the stack actions of Γ^{+-} are simply seen as output actions,
- the fail locations are reached in any of the following three cases: (i) an unspecified output (or stack) action is observed; (ii) an output (or stack) action occurs earlier or later than specified; (iii) a timeout occurs.

A *DTPTIO* has only one clock which is reset every time the tester observes an action. As a consequence, all the guards of the *DTPTIO* are satisfiable. We define the *DTPTIO* of a *TPAIO* in Def. 5.

Definition 5 (DTPTIO): The *DTPTIO* $T^T = (L^T, l_0^T, \Sigma_{out} \cup \Sigma_{in}, \Gamma, \{y\}, \Delta^T, F^T)$ of a *TPAIO* $T = \langle L, l_0, \Sigma_{out} \cup \Sigma_{in} \cup \{\tau\}, \Gamma, X, \Delta, F\rangle$ is a *TPAIO* with only one clock y that is a new clock w.r.t X where:

- $L^T \subseteq 2^{(L \times CC(X \cup \{y\}))} \cup \{fail\}$ is a set of symbolic locations,
- l_0^T is the initial symbolic location,
- $F^T \subseteq L^T$ is a set of accepting symbolic locations,
- $\Delta^T \subseteq L^T \times \Sigma_{out} \cup \Sigma_{in} \cup \Gamma^{+-} \times Grd(\{y\}) \times \{y\} \times L^T$ is a finite set of transitions.

C. Reachability Automaton of a DTPTIO Computation

A *TPAIO* does not only specify clock constraints but also stack constraints. Therefore, applying the algorithm of [9] for generating analog-clock tests from a *TPAIO* is not sufficient. It is necessary to take the stack content into account to avoid system deadlocks. For example, the pop action of a symbol that is not on the top of the stack would provoke a deadlock. We compute a reachability automaton for taking the stack constraints into account.

Let $\langle L^T, l_0^T, \Sigma_{out} \cup \Sigma_{in}, \Gamma, \{y\}, \Delta^T, F^T\rangle$ be a *DTPTIO*. We propose to compute a representation of its reachable locations from its initial location. This representation is called the *reachability automaton* of the *DTPTIO*. It is a finite automaton whose transition labels are sequences of transitions of the *DTPTIO* ($\in \Delta^{T^*}$). A π-transition $(l^T, \pi, l^{T'})$ is a transition that reaches the symbolic location $l^{T'}$ from l^T by taking the path π. We propose in Def. 6 the rules R_1 to R_5 that, applied repeatedly, define the *RA*.

Definition 6 (RA of a DTPTIO): The *RA* of a *DTPTIO* $\langle L^T, l_0^T, \Sigma_{out} \cup \Sigma_{in}, \Gamma, \{y\}, \Delta^T, F^T\rangle$ is the automaton $\langle L^R, l_0^T, (\Delta^T)^*, \Delta^R, F^T\rangle$ where $L^R = L^T \setminus \{fail\}$ and $\Delta^R \subseteq L^R \times (\Delta^T)^* \times L^R$ is the smallest relation that satisfies the following conditions. Let $t_1 \stackrel{def}{=} (l_1^T, a^+, g_1, \{y\}, l_2^T)$, $t_2 \stackrel{def}{=} (l_2^T, a^-, g_2, \{y\}, l_3^T)$, $t_3 \stackrel{def}{=} (l_3^T, a^-, g_3, \{y\}, l_4^T)$, $t_4 \stackrel{def}{=} (l_2^T, a^+, g, \{y\}, l_3^T)$ and $t_5 \stackrel{def}{=} (l_2^T, b^+, g_2, \{y\}, l_3^T)$ be five transitions in Δ^T where l_2^T, l_3^T, l_4^T are not the $fail$ location and $a \neq b$.

- $\mathbf{R}_1$: $(l^T, t_1, l^{T'}) \in \Delta^R$,
- $\mathbf{R}_2$: $(l^T, t, l^{T'}) \in \Delta^R$ if $t \stackrel{def}{=} (l^T, A, g, \{y\}, l^{T'})$ and $t \in \Delta^T$,
- $\mathbf{R}_3$: $(l_1^T, t_1.t_2, l_3^T) \in \Delta^R$,
- $\mathbf{R}_4$: $(l_1^T, t_1.\pi.t_3, l_4^T) \in \Delta^R$ if $(l_2^T, \pi, l_3^T) \in \Delta^R$, $\pi \neq t_4$ and t_1 is not a prefix of π and t_3 is not a suffix of π.
- $\mathbf{R}_5$: $(l_1^T, \pi_1.\pi_2, l_3^T) \in \Delta^R$ if $(l_1^T, \pi_1, l_2^T) \in \Delta^R$, $(l_2^T, \pi_2, l_3^T) \in \Delta^R$, $\pi_1 \neq t_1$, $\pi_2 \neq t_5$ and π_1 is not a prefix of π_2 and π_2 is not a suffix of π_1.

We have adapted an algorithm by Finkel et al. [10], that originally computes an *RA* from a *PA*, to compute an *RA* from a *DTPTIO*. Our modifications are as follows:

1) we compute the path of each transition. Any π-transition in the *RA* corresponds to the path π in the *DTPTIO*;
2) the problem addressed in [10] being to check the location reachability, the paths are ignored and there is only one transition between two symbolic locations l^T and $l^{T'}$. As we want to cover the transitions, we record as many transitions as required to cover all the transitions between the two symbolic locations;
3) we consider by means of the rule R_2, the transitions labelled by a symbol of $\Sigma_{out} \cup \Sigma_{in}$ in addition to the push and pop ones;
4) the reflexive transitions are not used in [10] because they don't cover any new state. By contrast in the rule R_5, we possibly extend an existing π-transition on its right or on its left by one occurrence of a reflexive transition, provided that it covers at least one new transition.

The computation of the paths is based on transition coverage. Adding a new π-transition $(l^T, \pi, l^{T'})$ is performed only if π covers a new transition between l^T and $l^{T'}$ of the *DTPTIO*. Thus our algorithm applies the rules R_1 to R_5 to compute the smallest set of transitions Δ^R that covers all the transitions.

D. Generating Nominal Behaviour Test Cases

We first define a tree of test cases in Def. 7.

Definition 7 (Tree of Test Cases): Let $T = \langle L, l_0, \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}, \Gamma, X, \Delta, F\rangle$ be a *TPAIO* specification model, and

$T^T = \langle L^T, l_0^T, \Sigma_{in} \cup \Sigma_{out}, \Gamma, \{y\}, \Delta^T, F^T \rangle$ be the *DTPTIO* of T. A tree of test case is a deterministic acyclic *TPAIO* whose locations are either $pass$, or $fail$, or $stack\_fail$, or symbolic configurations (l, c, p) where $l \in L$, $c \in CC(X \cup \{y\})$ and $p \in \Gamma^*$, defining a set of configurations (l, v, p) such that v satisfies c.

We propose, in order to produce a set of nominal test cases, to select the executions that go from the initial location to a final symbolic location with an empty stack. This is obtained from the *RA* by selecting its π-transitions going from the initial symbolic location l_0^T to a final one $l_f^T \in F^T$, with a label π different from $[(l_0^T, a^+, g_0, \{y\}, l_f^T)]$. The non-verdict nodes are a set of configurations (location, clock valuation, stack content) of the semantics of the *TPAIO*. To model the case where the pop of a symbol by the IUT is observed, although the symbol should not be on top of the stack according to the specification, we propose to add a special verdict $stack\_fail$.

V. Soundness and Coverage of the Method

This section discusses the soundness and coverage of our method for test generation from a *TPAIO*.

A. Soundness

Theorem 1: A symbolic location l_i^T is reachable in a *DTPTIO* if there exists a π-transition (l_0^T, π, l_i^T) in its *RA*.

Proof: The proof is by induction and by cases on each rule. The induction assumption is that the *RA* transitions that are merged into new transitions are sound. We prove this assumption to be true by proving that the rules R_1, R_2 and R_3, that create *RA* transitions only from *DTPTIO* ones, are sound. Then we prove that the rules R_4 and R_5 preserve that soundness.

- R_1 case: if there exists a transition $l_1^T \xrightarrow{l_1^T \xrightarrow{a^+, g_1, \{y\}} l_2^T} l_2^T \in \Delta^R$ then l_2^T is reachable from l_1^T in the *DTPTIO* because by definition of the *DTPTIO*, $\exists (l_1, v_1).((l_1, v_1) \in l_1^T$ and $v_1 \wedge g_1$ is satisfiable$)$.
- R_2 case: if there exists a transition $l_1^T \xrightarrow{l_1^T \xrightarrow{A, g_1, \{y\}} l_2^T} l_2^T \in \Delta^R$ then l_2^T is reachable from l_1^T in the *DTPTIO* because by definition of the *DTPTIO*, $\exists (l_1, v_1).((l_1, v_1) \in l_1^T$ and $v_1 \wedge g_1$ is satisfiable$)$.
- R_3 case: if there exists a transition $l_1^T \xrightarrow{l_1^T \xrightarrow{a^+, g_1, \{y\}} l_2^T \xrightarrow{a^-, g_2, \{y\}} l_3^T} l_3^T \in \Delta^R$ then l_3^T is reachable from l_1^T in the *DTPTIO* because it is always possible to pop a after a has been pushed and by definition of the *DTPTIO*, $\exists (l_1, v_1).((l_1, v_1) \in l_1^T$ and $v_1 \wedge g_1$ is satisfiable$)$ and $\exists (l_2, v_2).((l_2, v_2) \in l_2^T$ and $v_2 \wedge g_2$ is satisfiable$)$.
- R_4 case: if there exists a transition $l_1^T \xrightarrow{l_1^T \xrightarrow{a^+, g_1, \{y\}} l_2^T \xrightarrow{\pi} l_3^T \xrightarrow{a^-, g_3, \{y\}} l_4^T} l_4^T \in \Delta^R$ then l_4^T is reachable from l_1^T in the *DTPTIO* because l_3^T is reachable from l_2^T according to the induction hypothesis, and it is always possible to pop a after a has been pushed, and by definition of the *DTPTIO* $\exists (l_1, v_1).((l_1, v_1) \in l_1^T$ and $v_1 \wedge g_1$ is satisfiable$)$ and $\exists (l_3, v_3).((l_3, v_3) \in l_3^T$ and $v_3 \wedge g_3$ is satisfiable$)$.
- R_5 case: if there exists a transition $l_1^T \xrightarrow{l_1^T \xrightarrow{\pi_1} l_2^T \xrightarrow{\pi_2} l_3^T} l_3^T \in \Delta^R$ then l_3^T is reachable from l_1^T in the *DTPTIO* because l_2^T is reachable from l_1^T and l_3^T is reachable from l_2^T, according to the induction hypothesis.

■

Theorem 2: A location l_i is reachable in a *TPAIO* iff there exists a path that go from the initial symbolic location l_0^T to l_i^T in the *RA* of its *DTPTIO* where $(l_i, v_i) \in l_i^T$.

Proof: Let g be the guard of a transition (l, a, g, X', l'). A transition $(l^T, a, u, \{y\}, l'^T)$ is added to Δ^T where $a \in \Sigma \cup \Gamma^{+-}$ only if $\exists (l, v).((l, v) \in l^T$ and $v \wedge u \wedge g$ is satisfiable$)$, by property of the *DTPTIO*. This means that the construction of a *DTPTIO* from a *TPAIO* takes its clock constraints into account. As it leaves the stack ones unchanged, then Theorem 2 is a direct consequence of Theorem 1. ■

Our tests are correct in the sense that only the non conform executions are rejected.

Theorem 3: Let $\pi = S_0 \xrightarrow{a_0, g_0, \{y\}} S_1 \xrightarrow{a_1, g_1, \{y\}} ...S_{n-1} \xrightarrow{a_{n-1}, g_{n-1}, \{y\}} S_n \xrightarrow{a_n, g_n, \{y\}} S_{n+1}$ be a path of a test case of a specification $T = \langle L, l_0, \Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}, \Gamma, X, \Delta, F \rangle$ where $S_i \subseteq (L \times CC(X \cup \{y\}) \times \Gamma^*)$, $g_i \in Grd(\{y\})$, $a_i \in \Sigma_{out} \cup \Sigma_{in} \cup \Gamma^{+-} \cup \{-\}$ for each $0 \leqslant i \leqslant n$ and $l_{n+1} \in \{fail, stack\_fail\}$. If a verdict $fail$ or $stack\_fail$ is observed while executing the implementation I, then I does not conform to the specification T.

Proof: Let $\rho = \delta_0 a_0 \delta_1 a_1 \delta_2 ... \delta_{n-1} a_{n-1} \delta_n a_n \in RT(\Sigma_{in} \cup \Sigma_{out} \cup \Gamma^{+-})$ be the trace of the path π. S_n is the current symbolic location after the execution of $\delta_0 a_0 \delta_1 a_1 \delta_2 ... \delta_{n-1} a_{n-1} \delta_n$ and g_n is the coarsest partition (see [9]) computed such that $\delta_n \in g_n$. Let $l_n^T = \{(l_1, v_1), ..., (l_n, v_n)\}$ be the symbolic location of S_n where $S_n = \{(l_1, v_1, p_1), ..., (l_n, v_n, p_n)\}$. Reaching $fail$ or $stack\_fail$ is due to one of the following three cases:

- $fail$ occurs after an unspecified stack or output action a_n has been observed. If the set of transitions labelled by a_n whose guards are satisfied in l_n^T is empty, then the transition $(l_n^T, a_n, \{y\}, fail)$ is a transition of the *DTPTIO*. Therefore, $a_n \notin out(T\ after\ \delta_0 a_0 \delta_1 a_1 \delta_2 ... \delta_{n-1} a_{n-1} \delta_n)$, so I does not conform to T;
- $fail$ occurs after a timeout δ_n $(a_n = -)$ has been observed. Therefore, $\forall (l_i, v_i).((l_i, v_i) \in l_n^T \wedge v_i + \delta_n \notin out(T\ after\ \delta_0 a_0 \delta_1 a_1 \delta_2 ... \delta_{n-1} a_{n-1}))$, so I does not conform to T;
- $stack\_fail$ occurs after a pop action a_n has been observed, acceptable by the specification but in a context where the symbol cannot be on the top of the stack p_i of each state (l_i, v_i, p_i) in S_n. Therefore, I does not conform to T.

■

Thus for every non-conformance detected by a test case, there is a non conformance between the implementation and the specification (*TPAIO*).

B. Coverage

In Section IV-C, we have presented a method for computing an *RA* from a *DTPTIO*. The algorithm that computes the

RA takes the coverage of the transitions of the *DTPTIO* into account. It only adds a new π-transition $(l^T, \pi, l^{T'})$ when π covers a new transition not covered by the other π'-transitions $(l^T, \pi', l^{T'})$. The paths of all the π-transitions that reach a final symbolic location of the *RA* cover all the transitions of the *DTPTIO*. In our example, also all the reachable locations and transitions of the *TPAIO* are covered.

VI. Execution and Experimental Results

We define in this section a test execution on an implementation, and we show some experimental results.

A. Execution of Tests on Implementation

The execution of a test case TC on an implementation I can be defined as the parallel composition of I and TC. I does not conform to T if a location $(l, fail)$ where $l \in L^I$ is reachable in the parallel composition $I \parallel TC$. Let $PC = I \parallel TC$ be a *TPAIO*. We apply the following steps to check the reachability of a location $(l, fail)$ in PC:

1) determinization of PC, without adding the $fail$ locations and the transitions that lead to them, to obtain a tester PC^T;
2) computation of an *RA* PC^R from the PC^T;
3) verification if a location $(l, fail)$ is reachable in PC. This is obtained by means of Theorem 2 and Theorem 1 that formalize the link between PC, PC^T and PC^R: a location $(l, fail)$ is reachable in PC iff there exists a path that goes from the initial symbolic location l_0^T to a location l_i^T of PC^R (the RA of its *DTPTIO* PC^T), where $((l, fail), v_i) \in l_i^T$.

B. Experimental Results

We have built a proof-of-concept tool for generating tests from a non-deterministic *TPAIO*. It is written in java. We also have implemented the *TPAIO* mutation operators of Section III-B to automatically generate mutants of a *TPAIO*.

The experimentation has been performed on the example *TPAIO* of Figure 1. We got 68 mutants by applying the three mutation operators presented in Section III-B. The results of executing the tests that we have generated from the *TPAIO* appear in Table I. Column one is for the mutation operator applied. Column two tells how many mutants were obtained with it. Column three indicates the number of mutants unkilled, i.e. undetected as non-conform, while column four indicates the number of mutants killed, for which a non-conformance is pronounced.

Operator	Mutants	Unkilled	Killed
Change action	52	4	48
Change guard	8	1	7
Negate guard	8	0	8
Total	68	5	63

TABLE I. Experimental Results

This table shows that more than 90% (63/68) of the mutants have been killed by our method. Among the five unkilled mutants, two are in fact conform while three are not. Indeed, applying the mutation operator *Guard change* or *Action change* does not necessarily contradict the specification. This is the case with the two conform mutants. The non-conformance of the three others is not detected by our tree of test cases.

VII. Conclusion and Further Works

We have presented a test generation method from non-deterministic *TPAIO*. Our method proceeds by determinizing the *TPAIO* and then computing the reachable locations and transitions, to generate off-line tests that cover the reachable locations and transitions. The tester observes the stack and output actions, as well as the delays. For the determinization step, our method first adapts the *TAIO* tester computation method of [9] to the *TPAIO* case. We obtain a *DTPTIO* that is a one-clock *TPAIO*. Its locations are either symbolic ones, or defined as being rejecting ($fail$). In a second step, we adapt another algorithm presented in [10] for *PA*, for computing the *RA* of the *DTPTIO*. We compute the paths of the *DTPTIO* associated with each transition of the *RA*. The computation of the *RA* takes into account the coverage of all the transitions of the *TPAIO*. By using the paths of transitions of *RA* and the *DTPTIO*, a tree of test cases is generated. We have defined a mutation testing method for non-deterministic *TPAIO*, and showed with it on an example the efficiency of our method.

Some further experiments have to be performed to better analyse the efficiency of our method. As for the example treated in this paper, it already shows that our test suite could be improved to kill all the non-conform mutants. This could be obtained during the test execution computation by identifying earlier the fail locations reached, before the RA computation possibly makes them unreachable.

References

[1] P. A. Abdulla, M. F. Atig, and J. Stenman, "Dense-timed pushdown automata," in LICS, 2012, pp. 35–44.

[2] R. Alur and D. L. Dill, "A theory of timed automata," TCS, no. 2, 1994, pp. 183 – 235.

[3] J.-M. Autebert, J. Berstel, and L. Boasson, "Context-free languages and pushdown automata," in Handbook of Formal Languages. Springer, 1997, vol. 1, pp. 111–174.

[4] F. Cassez, J. J. Jessen, K. G. Larsen, J.-F. Raskin, and P.-A. Reynier, "Automatic synthesis of robust and optimal controllers — an industrial case study," in HSCC '09. Springer, 2009, pp. 90–104.

[5] A. Trivedi and D. Wojtczak, "Recursive timed automata," in ATVA'10, ser. LNCS, vol. 6252. Springer, 2010, pp. 306–324.

[6] M. Benerecetti, S. Minopoli, and A. Peron, "Analysis of timed recursive state machines," in TIME 2010. IEEE, 2010, pp. 61–68.

[7] A. Bouajjani, R. Echahed, and R. Robbana, "On the automatic verification of systems with continuous variables and unbounded discrete data structures," in Hybrid Systems II, ser. LNCS, vol. 999, 1995, pp. 64–85.

[8] R. Chadha, A. Legay, P. Prabhakar, and M. Viswanathan, "Complexity bounds for the verification of real-time software," in VMCAI'10, ser. LNCS, vol. 5944. Springer, 2010, pp. 95–111.

[9] M. Krichen and S. Tripakis, "Conformance testing for real-time systems," FMSD, vol. 34, no. 3, Jun. 2009, pp. 238–304.

[10] A. Finkel, B. Willems, and P. Wolper, "A direct symbolic approach to model checking pushdown systems (ext. abs.)," in Infinity, ser. ENTCS, vol. 9, 1997, pp. 27–37.

[11] B. Aichernig, F. Lorber, and D. Nickovic, "Time for mutants-model-based mutation testing with timed automata," in Tests and Proofs, ser. LNCS. Springer, 2013, vol. 7942, pp. 20–38.

[12] J. Tretmans, "Testing concurrent systems: A formal approach," in Concurrency Theory, ser. LNCS, vol. 1664. Springer, 1999, pp. 46–65.

[13] G. Sénizergues, "L(a) = l(b)? decidability results from complete formal systems." in ICALP, ser. LNCS, vol. 2380. Springer, 2002, pp. 1–37.