HAL
open science

Test Generation from Timed Pushdown Automata with Inputs and Outputs

Hana Hemdi, Jacques Julliand, Pierre-Alain Masson, Riadh Robbana

To cite this version:

Hana Hemdi, Jacques Julliand, Pierre-Alain Masson, Riadh Robbana. Test Generation from Timed Pushdown Automata with Inputs and Outputs. A-MOST 2015, 11th Workshop on Advances in Model Based Testing. Co-located with ICST 2015, 2015, Graz, Austria. pp.***–***. hal-01304670

HAL Id: hal-01304670

https://hal.science/hal-01304670

Submitted on 20 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Test Generation from Timed Pushdown Automata with Inputs and Outputs

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 Laboratory and INSAT, University of Carthage, Tunisia

***Abstract*—We consider in this paper the model of Timed Pushdown Automata with Inputs and Outputs (TPAIO), for which state reachability can only be solved in exponential time. We compute by means of a polynomial algorithm a reachability timed automaton (RTA), thus partial, of a TPAIO. When the algorithm is applied to untimed pushdown automata, the reachability is equivalent in both automata. But with the addition of clock constraints, reachability in the RTA is only a sufficient condition. To decide if a succession of timed transitions can be executed, we compute the backward closures of the clock constraints, and evaluate them by means of satisfiability decision procedures. Additionally, we compute a path table that relates a feasible transition of the RTA to the corresponding path of the TPAIO. We accept the incompleteness of our method as a price to pay for efficiency. It can be used in test generation since testing is incomplete by nature. Test generation relies on unfolding the transitions of the reachability timed automaton thanks to the path table.**

Keywords: Timed Pushdown Automata; Reachability Timed Automata; Clock Constraints Backward Closure; Test Generation from Automata; Conformance Relation for TPAIO.

I. Introduction

Systems are commonly modelled by various types of transition systems including finite automata, pushdown automata (*PA*), timed automata (*TA*), etc. The verification of these systems, as well test generation from their models, are very active research areas[1], [2], [3].

PA [4] are equipped with a stack, and can model recursive systems. The reachability problem is the problem of deciding whether an automaton can reach a particular location from an initial location. This problem is decidable [5][6]. *TA* were introduced by Alur and Dill [7], and have become a standard modelling formalism for real-time systems. They are equipped with a finite set of real-valued clocks, in which constraints on the clocks are used to restrict the behaviours of the automata. One of the most basic problems in *TA* is the location reachability problem. Reachability in *TA* is a decidable problem [8].

In this paper we consider *TPA*, i.e. *TA* equipped with a stack. with inputs and outputs to model for example recursive procedure calls in real time systems. Theoretically, the reachability problem is solved [9], [10]. Although reachability in *PA* can be verified in polynomial time, adding clocks provokes an exponential blow up in complexity (see for example [11]), so that reachability in *TPA* can only be verified in exponential time. Besides, the usual approach to deal with *TA*, that are space infinite due to the infinite domains of the clocks, is to perform a region graph partitioning. This provides a finite representation of the *TA*, but the number of regions grows exponentially with the number of clocks. The use of ϵ-transitions and backward closures of clock constraints, as in [12], allows to express the successive constraints as a SAT problem over the clock constraints.

In this paper, we propose an approach for computing tests from the *TPAIO* model. Following [5], we propose a set of rules to build a reachability timed automaton (*RTA*) from a *TPA*. The *RA* is computed in [5] from a *PA* in polynomial time, and location reachability is equivalent in both models. In our case, we have adapted the rules to take clock constraints into account in addition to stack ones. Applying these rules until saturation would theoretically result in computing an *RTA* where location reachability is equivalent to that of the *TPA*. But not only the computation time becomes exponential, it may also not terminate. Hence our proposition is an incomplete method. We propose an algorithm that applies the rules with a termination criterion based on transitions coverage. It operates in polynomial time and is guaranteed to terminate. Reachability in the *RTA* becomes only a sufficient condition to reachability in the *TPA*. While in verification completeness is usually a must, test generation can deal with some incompleteness, as software testing is an incomplete activity by nature. By applying to a *TPA* a polynomial transformation into a *RTA*, and with the use of SMT solvers to solve the backward closures of clock constraints, our method can efficiently generate a set of tests from a *TPAIO*. To summarize, our contributions are to: (i) define *tpioco*: a conformance relation for the *TPAIO* model; (ii) adapt the reachability computation of [5] to the case of *TPAIO* in the goal to generate tests; (iii) define a method that is incomplete but polynomial to compute a partial *RTA* of a *TPA*; (iv) generate test cases by covering the reachable locations and transitions of the *TPAIO*. To our knowledge, these problems solved for the *TA* and the *PA* had not been handled for the *TPA* yet.

The paper is organized as follows. Section II presents the *TA*, the conformance relation $tioco$ and the backward closure in *TA*. Section III presents our *TPAIO* model, a conformance relation for *TPAIO* and an illustrative example of a *TPAIO*. Section IV applies transition merging to *TPA* and gives rules to define a complete *RTA* of a *TPA*. It also presents the polynomial algorithm to compute a partial *RTA* and a path table from a *TPA*. Our method for generating tests from a *TPAIO* and a conformance relation is presented in Section V. In Section VI, we illustrate the soundness, incompleteness and test coverage of our method. We conclude and indicate future work in Section VII.

II. Background

This section defines *TA*, the timed input-output conformance relation and the backward closure in *TA*.

A. Timed Automata

Let $Grd(X)$ be the language of clock guards defined by the following grammar $g ::= x \sim n \mid g \wedge g \mid g \vee g \mid \mathsf{true} \mid \mathsf{false}$ where $\sim \in \{<, \leq, >, \geq, =\}$, X is a set of clocks, x is a clock in X taking its values in $\mathbb{R}^+$ and n is a constant in $\mathbb{N}$.

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 $\langle S^T, s_0^T, \Delta^T\rangle$ whose states in S^T are pairs $(l, v) \in L \times (X \to \mathbb{R}^+)$ where l is a location and v is 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 where each clock is augmented by the delay 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.$\sigma = (l_0, v_0) \to^{\delta_0} (l_0, v_0 + \delta_0) \to^{a_0} (l_1, v_1) \to^{\delta_1} (l_1, v_1 + \delta_1) \to^{a_1} (l_2, v_2) \to^{\delta_2} \ldots \to^{a_{n-1}} (l_n, v_n)$ 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 $v_i \models g_i$ for $0 \leqslant i < n$. A run alternates timed and discrete transitions. Its trace is $\rho = \delta_0 a_0 \delta_1 a_1 \ldots \delta_n a_n$, a finite sequence 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 a trace that is the projection of a trace ρ on Σ_1 with preserved delays and where $\Sigma_1 \subseteq \Sigma$. For example, if $\rho = 5a4b2$, then, $P_{\{a\}}(\rho) = 5a42 = 5a6$. $Time(\rho)$ is the sum of all the delays in 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 if there exists a run σ from s_0^T to s such that ρ is its trace. $s_0^T \to^\rho$ 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 distinction 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 the commands of Σ_{in} and observes the output of Σ_{out}. The implementation under test (IUT), sends the observable actions of Σ_{out} and accepts the commands of Σ_{in}.

Let $\Sigma = \Sigma_{in} \cup \Sigma_{out}$ and $\Sigma_\tau = \Sigma \cup \{\tau\}$. A *TAIO* is deterministic if for all location l in L, for all action a in Σ_τ and for all couple 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. The set of reachable states of a *TAIO* T, denoted $Reach(T)$, is the set: $\{s^T \in S^T \mid \exists \rho.(\rho \in RT(\Sigma) \wedge s_0^T \to^\rho s^T\}$. 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* [1]. $tioco$ is an extension of the *ioco* relation of Tretmans [2]. The main difference between $tioco$ and $ioco$ is that $ioco$ uses the notion of quiescence. In [1], the $tioco$ relation doesn't use quiescence because the timeouts are explicitly specified. The assumptions are that the specification of the system to be tested is a non-blocking *TAIO*, and that its 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 $\rho \in RT(\Sigma)$:

- 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 all states of T reachable 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 all delays which 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: Timed Input-Output Conformance Relation): 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

$$I \ tioco \ T \text{ iff } \forall \rho.(\rho \in ObsTTraces(T) \implies out(I \ after \ \rho) \subseteq out(T \ after \ \rho)).$$

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.

C. Backward Closure in Timed Automata

As in [5], our method for computing location reachability introduces ϵ-transitions. An ϵ-transition in a *PA* goes from a location to another without modifying the stack content. It is used to represent by means of a single transition a succession of push and pop ones that leave the stack unchanged at the end. Our intention is to merge a sequence of consecutive ϵ-transitions in the *TPA* case. The successive clock constraints have to be accumulated and their verification shifted backward to the beginning of the sequence. We use for this the backward closure of constraints as in [12].

Definition 3 (Backward closure of a constraint): Let g be a clock constraint and X be a set of clocks. The backward

closure of g on X, denoted by $\overleftarrow{g}^X$, is a formula that is satisfied by a clock valuation v if g will be satisfiable after the clocks of X have been reset, and a delay δ has passed:

$$v \vDash \overleftarrow{g}^X \text{ if } \exists \delta \cdot (\delta \geq 0 \wedge v[X := 0] + \delta \vDash g).$$

III. *TPAIO* and Conformance Relation

In this section, we first define the *TPAIO* model, then a conformance relation for a *TPAIO*. We also present an example of a *TPAIO* that models a recursive program.

A. Timed Pushdown Automata with Inputs and Outputs

A *TPA* $T = \langle L, l_0, \Sigma, \Gamma, X, \Delta, F \rangle$ is a *TA* equipped with a stack. Its operational semantics is a transition system $< S^T, s_0^T, \Delta^T >$ where the locations called states are configurations made of three components (l, v, p) where l is a location of the *TPA*, v is a clock valuation in $X \to \mathbb{R}^+$ and p is a stack content in Γ^*. In this paper, we consider *TPA* with Inputs and Outputs (*TPAIO*).

Definition 4 (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, $\Sigma = \Sigma_{in} \cup \Sigma_{out}$ where Σ_{in} is a finite set of input actions, Σ_{out} is a finite set of output actions, Γ is a stack alphabet ($\Sigma_{out} \cap \Sigma_{in} = \varnothing$, $\Sigma_{in} \cap \Gamma = \varnothing$ and $\Sigma_{out} \cap \Gamma = \varnothing$), 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 $(l, v, p) \to^\delta (l, v + \delta, p)$. For a transition (l, act, g, X', l'), there are three types of discrete transitions when v satisfies g: (1) *push* when $act = a^+$: $(l, v, p) \to^{a^+} (l', v[X' := 0], p.a)$ where $a \in \Gamma$, (2) *pop* when $act = a^-$: $(l, v, p.a) \to^{a^-} (l', v[X' := 0], p)$ where $a \in \Gamma$, (3) *output or input* when $act = A \in \Sigma$: $(l, v, p) \to^A (l', v[X' := 0], p)$. A *TPAIO* is normalized if it executes separately push and pop operations. All *TPAIO* can be normalized since all *PA* can be normalized [13]. In the remainder of the paper, we consider that *TPAIO* are always normalized deterministic *TPAIO* and we denote a the actions of Γ, and A the actions of Σ.

We define our conformance relation denoted *tpioco* for the *TPAIO* as an extension of the conformance relation *tioco*[1]. It is the same relation as $tioco$ for *TAIO* considering that the whole alphabet is $\Sigma \cup \Gamma^{+-}$ instead of $\Sigma \cup \{\tau\}$ (there is no observable action τ), the output alphabet is $\Sigma_{out} \cup \Gamma^{+-}$ instead of Σ_{out} and the input alphabet remains Σ_{in}.

B. Modelling of Recursive Programs

Figure 1 shows a *TPAIO* that is an abstraction of a function called pow that computes x^n. The location labels correspond to control point in the body of the function between each atomic instruction. pow^+ is a recursive call (push). pow^- is a return (pop) of a recursive call. Thus, $\Gamma = \{pow\}$. The atomic instructions and conditions are abstracted by the letters A to H so: $A \stackrel{def}{=}$ int res, $B \stackrel{def}{=} n = 0$, $C \stackrel{def}{=} n \neq 0$, $D \stackrel{def}{=}$ return 1, $E \stackrel{def}{=} n \bmod 2 = 0$, $F \stackrel{def}{=}$ return $res * res$, $G \stackrel{def}{=} n \bmod 2 \neq 0$ and $H \stackrel{def}{=}$ return $res * res * x$. All executions of atomic instructions are in Σ_{in}. All executions of conditions are in Σ_{out}. Thus, $\Sigma_{in} = \{A, D, F, H\}$ and $\Sigma_{out} = \{B, C, E, G\}$. We use the notation $!act$ to denote act as an output action of Σ_{out} and $?act$ to denote act as an input action of Σ_{in}.

IV. *RTA* Definition and Computation

From a *TPA*, we define a Reachability Timed Automaton (*RTA*) that is a *TA* whose the set of transitions can be infinite and each of them is labelled by ϵ. We extend for that a set of rules for *PA* issued from [5] to the *TPA* case, by taking the clock constraints into consideration. The sequences made of successive push and pop transitions are merged into ϵ-transitions. Then, according to Def. 5, the successive resulting ϵ-transitions are merged, etc. The clock constraints are accumulated during this process by computing their backward closure. This preserves reachability as proved by Lemma 1. Let $GrdB(X)$ the language of clock guards with backward closures defined by the following grammar $gb ::= g \mid g \wedge b$ where $g \in Grd(X)$, $b ::= \overleftarrow{g}^{X'} \mid \overleftarrow{gb}^{X'}$ and X' is a subset of clocks of X.

Definition 5 (Merging of two successive merged transitions): The successive transitions:

$$t_{1k} = l_1 \xrightarrow{\epsilon, g_1 \wedge g_2 \wedge \ldots \wedge \overleftarrow{g_{k-1}}^{X_{k-2}}, X_{k-1}} l_k \quad \text{and}$$

$$t_{kn} = l_k \xrightarrow{\epsilon, g_k \wedge g_{k+1} \wedge \ldots \wedge \overleftarrow{g_{n-1}}^{X_{n-2}}, X_{n-1}} l_n$$

are merged into the ϵ-transition $t_{1n} =$

$$l_1 \xrightarrow{\epsilon, g_1 \wedge g_2 \wedge \ldots \wedge g_{k-1} \wedge g_k \wedge g_{k+1} \wedge \ldots \wedge \overleftarrow{g_{n-1}}^{X_{n-2}}, X_{n-1}} l_n$$

where $k \geqslant 2$ and $n \geqslant k + 1$.

For example, the case where $k = 2$ and $n = 3$ merges the two ϵ-transitions $l_1 \xrightarrow{\epsilon, g_1, X_1} l_2 \xrightarrow{\epsilon, g_2, X_2} l_3$ into the ϵ-transition $l_1 \xrightarrow{\epsilon, g_1 \wedge \overleftarrow{g_2}^{X_1}, X_2} l_3$.

Lemma 1: Let t_{1k}, t_{kn} and t_{1n} be the transitions as defined in Def. 5. The location l_n is reachable from l_1 by applying successively the transition t_{1k} and the transition t_{kn} iff l_n is reachable from l_1 by applying the transition t_{1n}.

Proof: We first prove the left to right implication. We assume that the location l_n is reachable from the location l_1 by applying successively t_{1k} and t_{kn}. This means that there exists a valuation v_1 and a succession of delays $\delta_1, \delta_2, ..., \delta_{k-1}, ..., \delta_{n-1}$ such that there exists the following two runs: $\sigma_{1k} = (l_1, v_1) \to^{\delta_1} (l_1, v_1 + \delta_1) \to^\epsilon (l_2, v_2) \to^{\delta_2} (l_2, v_2 + \delta_2) \to^\epsilon (l_3, v_3) \to \cdots \to^{\delta_{k-1}} (l_{k-1}, v_{k-1} + \delta_{k-1}) \to^\epsilon (l_k, v_k)$ and $\sigma_{kn} = (l_k, v_k) \to^{\delta_k} (l_k, v_k + \delta_k) \to^\epsilon (l_{k+1}, v_{k+1}) \ldots \to^{\delta_{n-1}} (l_{n-1}, v_{n-1} + \delta_{n-1}) \to^\epsilon (l_n, v_n)$ where $v_i = (v_{i-1} + \delta_{i-1})[X_{i-1} := 0]$ for $0 < i \leq n$.

Under this assumption, the transition t_{1n} is fireable as the values $v_1, \delta_1, \delta_2, ..., \delta_{n-1}$ make its guard satisfiable. Thus l_n is reachable by the run that is the concatenation of σ_{1k} and σ_{kn}. Proof of the right to left implication is similar.

■

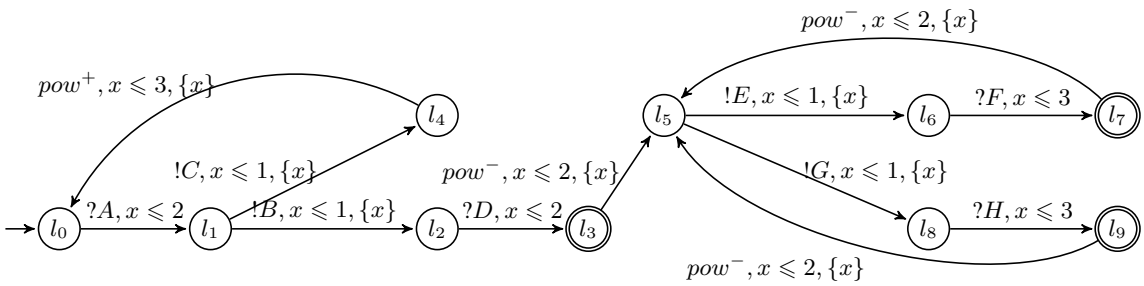

Figure 1. Example of a timed pushdown automaton

A. Complete RTA Definition Rules

Let $\langle L, l_0, \Sigma, \Gamma, X, \Delta, F\rangle$ be a *TPA*. We propose to define the *Reachability Timed Automaton* (*RTA*) of the *TPA*. It is a *TA* whose $\Sigma = \{\epsilon\}$. An ϵ-transition $l \xrightarrow{\epsilon, g, X'} l'$ is a transition that reaches the location l' from the location l without modifying the stack content. The reachable locations of the *RTA* are those that are reachable from its initial location by an ϵ-transition whose guard is satisfiable. We propose in Def. 6 the rules RA_1 to RA_4 that, applied repeatedly, define a *RTA*.

Definition 6 (RTA of a TPA): The *RTA* of a *TPA* $\langle L, l_0, \Sigma, \Gamma, X, \Delta, F\rangle$ is the *TA* $\langle L, l_0, \{\epsilon\}, X, \Delta^R, F\rangle$ where $\Delta^R \subseteq L \times \{\epsilon\} \times GrdB(X) \times 2^X \times L$ is the relation that satisfies the rules given in Table I.

Lemma 2: The transitions of the *RTA* that result from the rules RA_1 to RA_4 are fireable iff the *TPA* transitions that they merge are fireable.

Proof: The proof is by induction and by cases on each rule. The induction assumption is that the *RTA* transitions are sound before they are merged into new transitions. We prove this assumption to be true by proving that the rules RA_1 and RA_2, that create *RTA* transitions only from *TPA* ones, are sound. Then we prove that the rules RA_3 and RA_4 preserve that soundness.

- RA_1 case: $l_1 \xrightarrow{A, g_1, X_1} l_2 \in \Delta$ is fireable if there exists a clocks valuation v_1 such that $v_1 \models g_1$. Thus the transition $l_1 \xrightarrow{\epsilon, g_1, X_1} l_2 \in \Delta^R$ is also fireable from v_1. This is obviously true in the opposite direction.
- RA_2 case: first, regarding the stack constraints, the transitions are successively fireable because it is always possible to pop a after the label a has been pushed. As for the clock constraints, $l_1 \xrightarrow{a^+, g_1, X_1} l_2 \in \Delta$ and $l_2 \xrightarrow{a^-, g_2, X_2} l_3 \in \Delta$ are fireable if there exists a clocks valuation v_1 and a delay δ_2 such that $v_1 \models g_1$ and $v_1[X_1 := 0] + \delta_2 \models g_2$. These are exactly the conditions for which the transition $l_1 \xrightarrow{\epsilon, g_1 \wedge \overleftarrow{g_2}^{X_1}, X_2} l_3 \in \Delta^R$ is fireable, i.e. $v_1 \models g_1 \wedge \overleftarrow{g_2}^{X_1}$. This condition is equivalent to $v_1 \models g_1$ and $v_1 \models \overleftarrow{g_2}^{X_1}$. From Def. 3, $v_1 \models \overleftarrow{g_2}^{X_1}$ is satisfied if $\exists \delta_2.(\delta_2 \geqslant 0 \wedge v_1[X_1 := 0] + \delta_2 \models g_2)$ is satisfied.
- RA_3 case: the stack constraints are satisfied for the same reasons as in the previous case, because an ϵ-transition leaves the stack content unchanged. The clock constraints are also satisfied for the following reasons: the sequence of three transitions $l_1 \xrightarrow{a^+, g_1, X_1} l_2 \in \Delta$, $l_2 \xrightarrow{\epsilon, g_2 \wedge g_3 \wedge \ldots \wedge \overleftarrow{g_{k-1}}^{X_{k-2}}, X_{k-1}} l_k \in \Delta^R$ and $l_k \xrightarrow{a^-, g_k, X_k} l_{k+1} \in \Delta$ are fireable if there exists $v_1, \delta_2, ..., \delta_k$ such that $v_1 \models g_1$ and $v_2 = v_1[X_1 := 0]$, $v_2 + \delta_2 \models g_2$ and $v_3 = (v_2 + \delta_2)[X_2 := 0]$, $v_k + \delta_k \models g_k$. For such values the transition $l_1 \xrightarrow{\epsilon, g_1 \wedge g_2 \wedge g_3 \wedge \ldots \wedge g_{k-1} \wedge \overleftarrow{g_k}^{X_{k-1}}, X_k} l_{k+1} \in \Delta^R$ is fireable because its guard defined by Def. 3 is satisfied. From Def. 3 this condition is the following $v_1 \models g_1 \wedge g_2 \wedge \overleftarrow{g_3}^{X_2} \equiv v_1 \models g_1 \wedge \exists \delta_2.(\delta_2 \geqslant 0 \wedge v_1[X_1 := 0] + \delta_2 \models g_2 \wedge \overleftarrow{g_3}^{X_2} \equiv v_1 \models g_1 \wedge \exists \delta_2.(\delta_2 \geqslant 0 \wedge v_1[X_1 := 0] + \delta_2 \models g_2 \wedge \exists \delta_3.(\delta_3 \geqslant 0 \wedge (v_1[X_1 := 0] + \delta_2)[X_2 := 0] + \delta_3 \models g_3))$. Our assumption is exactly this condition: there exists v_1, δ_2, δ_3 such that $v_1 \models g_1$ and $v_2 = v_1[X_1 := 0]$ and $v_2 + \delta_2 \models g_2$ and $v_3 = (v_2 + \delta_2)[X_2 := 0]$ and $v_3 + \delta_3 \models g_3$.
- RA_4 case: it is a direct consequence of Lemma 1 as the rule RA_4 is the rule of Def. 5.

■

Remark 1: The rule RA_4 is the merging rule of Def. 5. Due to the rules RA_3 and RA_4, the repeated application of these rules may not converge in the case where a cycle of ϵ-transition is created on a location l. Merging this cycle with another ϵ-transition that enters (or leaves) l from (or towards) a location l' creates a new ϵ-transition between these two locations, that still can be merged again with the cycle, and so on… Such a cycle is satisfiable or not in terms of clock constraints. This satisfiability does not depend on the number of times the cycle is taken. Thus, to ensure its termination, an algorithm applying these rules repeatedly should take care of not taking a cycle of ϵ-transition more than once. Our algorithm of Fig. 3 in the next section takes this care.

B. Algorithm to Compute a Finite Partial RTA

We present an algorithm that applies finitely the rules RA_1 to RA_4 for building a finite partial *RTA* from a *TPA*. It is an adaptation of the algorithm [5], which originally computes a reachability automaton from a *PA*. The principle is to first gather into single ϵ-transitions the successive push and pop transitions, and then to incrementally explore how these ϵ-transitions can be combined to each other and to the remaining transitions. We extend this algorithm to the *TPA* case. The ϵ-transitions are merged the same way w.r.t. the stack constraints,

RA$_1$	$l_1 \xrightarrow{\epsilon, g_1, X_1} l_2 \in \Delta^R$ if $l_1 \xrightarrow{A, g_1, X_1} l_2 \in \Delta$
RA$_2$	$l_1 \xrightarrow{\epsilon, g_1 \wedge \overleftarrow{g_2}^{X_1}, X_2} l_3 \in \Delta^R$ if $l_1 \xrightarrow{a^+, g_1, X_1} l_2 \in \Delta$ and $l_2 \xrightarrow{a^-, g_2, X_2} l_3 \in \Delta$
RA$_3$	$l_1 \xrightarrow{\epsilon, g_1 \wedge g_2 \wedge g_3 \wedge \ldots \wedge g_{k-1} \wedge \overleftarrow{g_k}^{\overleftarrow{\ldots}^{\overleftarrow{X_{k-1}}^{\overleftarrow{\ldots}^{\overleftarrow{X_2}^{X_1}}}}}, X_k} l_{k+1} \in \Delta^R$ if $l_1 \xrightarrow{a^+, g_1, X_1} l_2 \in \Delta, l_2 \xrightarrow{\epsilon, g_2 \wedge g_3 \wedge \ldots \wedge \overleftarrow{g_{k-1}}^{\overleftarrow{\ldots}^{\overleftarrow{X_{k-2}}^{X_2}}}, X_{k-1}} l_k \in \Delta^R$ and $l_k \xrightarrow{a^-, g_k, X_k} l_{k+1} \in \Delta$
RA$_4$	$l_1 \xrightarrow{\epsilon, g_1 \wedge g_2 \wedge \ldots \wedge g_{k-1} \wedge g_k \wedge g_{k+1} \wedge \ldots \wedge \overleftarrow{g_{n-1}}^{\overleftarrow{X_{n-2}}^{\overleftarrow{\ldots}^{\overleftarrow{X_k}^{\overleftarrow{X_{k-1}}^{\overleftarrow{X_{k-2}}^{\overleftarrow{\ldots}^{\overleftarrow{X_1}}}}}}}}, X_{n-1}} l_n \in \Delta^R$ if $l_1 \xrightarrow{\epsilon, g_1 \wedge g_2 \wedge \ldots \wedge \overleftarrow{g_{k-1}}^{\overleftarrow{\ldots}^{\overleftarrow{X_{k-2}}^{\overleftarrow{\ldots}^{\overleftarrow{X_1}}}}}, X_{k-1}} l_k \in \Delta^R$ and $l_k \xrightarrow{\epsilon, g_k \wedge g_{k+1} \wedge \ldots \wedge \overleftarrow{g_{n-1}}^{\overleftarrow{X_{n-2}}^{\overleftarrow{\ldots}^{\overleftarrow{X_k}}}}, X_{n-1}} l_n \in \Delta^R$

TABLE I. *RTA* Building Rules

but we have additional rules for computing their time constraint backward closure at merging time. The resulting ϵ-transitions carry a guard w.r.t. the clocks. The satisfiability evaluation of these guards is postponed to a second phase, when all the merging have been performed. Additionally, our algorithm computes a path table, which associates each transition of the *RTA* with one or many paths of the *TPA*. Our modifications of the algorithm of [5] are summarized as follows.

1) We compute not only the ϵ-transitions in the *RTA* but also their paths. Any ϵ-transition in the *RTA* corresponds to one or many paths in the *TPA*.
2) Because the problem addressed in [5] is to check the locations reachability, the redundant ϵ-transition between two locations l and l' are not recorded in the result, although they have been computed. We record them as alternative possibilities for the clock constraints to be satisfiable.
3) We add the rule RA_1 because these transitions may carry some clock constraints that we cannot ignore, contrarily to the context of *PA* without clock constraint considered by Finkel et al.
4) The reflexive ϵ-transitions are not used in [5] to extend (on their right or on their left) the existing ϵ-transitions, because they do not change anything regarding accessibility. There again, we cannot ignore them due to their clock constraints.

To ensure the termination of our algorithm (see Remark 1), a new transition (l, ϵ, g, X, l') is added only if its path covers a new transition of the *TPA* between the locations l and l'.

The algorithm is given in Fig. 3. It computes the transitions of Δ^R in the table *paths*. Its input is a *TPA*. It returns a path table which associates each transition of the *RTA* with a set of paths of the *TPA*. To present this algorithm, we define the type **PATHS** = $Seq(\Delta)$ that is a sequence of transitions of the *TPA*, and the type **PATH_TABLES** =$\Delta^R \rightarrowtail$ set of $PATHS$ that is a surjective function that maps a set of paths to each transition of the *RTA*. This algorithm computes the transitive closure of ϵ-transitions only once by storing information in the data structures C_Direct and C_Trans on how the ϵ-transitions can be obtained. The algorithm enumerates all the possible pairs of locations, and searches for each of them if it can be exploited to form an ϵ-transition of the *RTA*.

The algorithm is in two steps: an initialization step from lines 1 to 16, and a processing step from line 17 to line 46. A *stack* is used to store the ϵ-transitions that have been encountered, but not yet exploited in conjunction with the other transitions. This stack is initialized in lines 1-3 by pushing onto it all the transitions labelled in Σ of the *TPA*, and all the trivial transitions (l, l) in lines 4-5. The two structures C_Direct and C_Trans are initially empty (lines 7-9). The C_Direct structure associates a set to each possible transition. It is used to apply the rules RA_2 and RA_3.

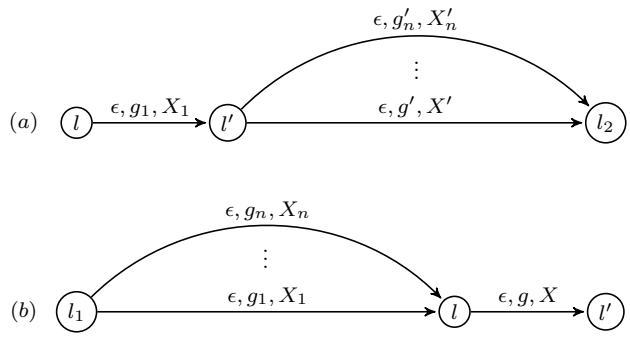

Figure 2. Possibilities of computing an ϵ-transition

For each pair of locations (l, l'), the set $C\_Direct(l, l')$ is initialized by a sequence of two transitions: a push transition a^+ and a pop transition a^- (lines 10-12). The C_Trans structure associates a set to each possible pair of transitions. It is used to apply the rule RA_4. Its initialization is performed by lines 13-16. In its second step (lines 17-46), the algorithm processes each transition popped off the stack, and determines its consequences when considering C_Direct and C_Trans. For an ϵ-transition and its path π between the locations l and l', the algorithm examines the two possibilities for computing other ϵ-transitions:

- By using $C\_Direct((l, l'))$ (line 33-35): for every $((l_1, l_2), [(l_1, a^+, g_1, X_1, l), (l', a^-, g', X', l_2)])$ in $C\_Direct((l, l'))$, our algorithm adds a new ϵ-transition between l_1 and l_2 where its path is $[(l_1, a^+, g_1, X_1, l)\hat{}\pi\hat{}(l', a^-, g', X', l_2)]$ where $t\hat{}\pi$ denotes the concatenation of the transition t with the path π.
- By using $C\_Trans((l, l'))$ (line 36-44): for every $((l_1, l_2), (l_3, l_4))$ in $C\_Trans((l, l'))$: for each path π_1 that forms an ϵ-transition between l_1 and l_2: our algorithm adds an $\epsilon-$transition as illustrated in

INPUT: A *TPA* $\langle L, l_0, \Sigma, \Gamma, X, \Delta, F\rangle$
OUTPUT: $paths \in PATH\_TABLE$
VARIABLES: $stack \in L \times L \leftrightarrow PATHS$; $C\_Direct \in L \times L \rightarrowtail$ set of $L \times L \leftrightarrow PATHS$; $C\_Trans \in L \times L \rightarrowtail$ set of $L \times L \rightarrowtail L \times L$; $l, l', l_1, l_2, l_3, l_4 \in L$; $\pi, \pi_1 \in PATHS$; $tr \in \Delta^R$; $t \in \Delta$; $coveredTransitions \in L \times L \rightarrowtail$ set of Δ

BEGIN
1: **for** every transition $(l, A, g, X, l') \in \Delta$ where $A \in \Sigma$ **do**
2: push $((l, l'), [(l, A, g, X, l')])$ on $stack$ /* implements the rule RA_1*/
3: **end for**
4: **for** every location $l \in L$ **do**
5: push $((l, l), \varnothing)$ on $stack$
6: **end for**
7: **for** every pair $(l, l') \in (L \times L)$ **do**
8: $C\_Direct(l, l') \leftarrow \varnothing$; $C\_Trans(l, l') \leftarrow \varnothing$; $coveredTransitions(l, l') \leftarrow \varnothing$
9: **end for**
10: **for** every pair $(l_1 \xrightarrow{a^+, g_1, X_1} l_2, l_3 \xrightarrow{a^-, g_3, X_3} l_4) \in (\Delta \times \Delta)$ where $a \in \Gamma$ **do**
11: $C\_Direct(l_2, l_3) \leftarrow C\_Direct(l_2, l_3) \cup \{((l_1, l_4), [(l_1, a^+, g_1, X_1, l_2), (l_3, a^-, g_3, X_3, l_4)])\}$
12: **end for**
13: **for** every triplet $(l, l', l'') \in (L \times L \times L)$ **do**
14: $C\_Trans(l, l') \leftarrow C\_Trans(l, l') \cup \{((l', l''), (l, l''))\}$
15: $C\_Trans(l, l') \leftarrow C\_Trans(l, l') \cup \{((l'', l), (l'', l'))\}$
16: **end for**
17: **while** $stack \neq emptyStack$ **do**
18: $((l, l'), \pi) \leftarrow pop(stack)$
19: $tr \leftarrow MergeTransitions(\pi)$
20: **if** $isNewTransitions(\pi, coveredTransitions((l, l')))$ **then**
21: **if** $\pi \neq []$ **then**
22: **if** $tr \notin dom(paths)$ **then**
23: $paths(tr) \leftarrow \{\pi\}$
24: **else**
25: $paths(tr) \leftarrow paths(tr) \cup \{\pi\}$
26: **end if**
27: **end if**
28: **for** t in π **do**
29: **if** $t \notin coveredTransitions((l, l'))$ **then**
30: $coveredTransitions((l, l')) \leftarrow coveredTransitions((l, l')) \cup \{t\}$
31: **end if**
32: **end for**
33: **for** $((l_1, l_2), [(l_1, a^+, g_1, X_1, l), (l', a^-, g', X', l_2)])$ in $C\_Direct((l, l'))$ **do**
34: $push\ ((l_1, l_2), (l_1, a^+, g_1, X_1, l)\hat{}\pi\hat{}(l', a^-, g', X', l_2)$ on $stack$ /* where $t\hat{}\pi$ denote the concatenation of t and π */
35: **end for**
36: **for** $((l_1, l_2), (l_3, l_4))$ in $C\_Trans((l, l'))$ **do**
37: **for** each π_1 in $getPaths(l_1, l_2, paths)$ **do**
38: **if** $l' = l_1$ **then**
39: $push\ ((l, l_2), \pi\hat{}\pi_1)$ on $stack$
40: **else**
41: $push\ ((l_1, l'), \pi_1\hat{}\pi)$ on $stack$
42: **end if**
43: **end for**
44: **end for**
45: **end if**
46: **end while**

END.

Figure 3. finite partial *RTA* computation algorithm

Fig. 2(a) between l and l_2, by extension on the left, when $l_1 = l'$ and as illustrated in Fig. 2(b) between l_1 and l', by extension on the right, when $l_2 = l$.

Our algorithm presented in Fig. 3 uses the following functions: (i) $MergeTransitions(\pi)$ that returns an ϵ-transition and merges all the transitions of its input path π, (ii) $isNewTransitions(\pi, \Delta)$ that verifies if the sequence of transitions π contains a transition which is not in the set of transitions Δ and (iii) $getPaths(l, l', paths)$ that returns the subset of paths in *paths* that lead from l to l'. To ensure the termination of our algorithm, the adding of a new transition (l, ϵ, g, X, l') is performed only if its path covers a new transition. Our algorithm uses the data structure $coveredTransitions$ that associates a set of transitions to a pair of locations. For every new transition between two locations l and l', $coveredTransitions$ stores the new covered transitions between l and l' (lines 29-32).

The algorithm of [5] operates in $O(n^3)$ where n is the number of locations of the *PA*. As a *TPA* contains output transitions that our algorithm treats differently than that of [5], and as the termination of our algorithm depends on the number of transitions between two locations, our algorithm is still polynomial.

Example 1: Table II shows some transitions and their paths from the initial location l_0 to a final one l_3, l_7 or l_9 of the *RTA* of the *TPA* of Fig. 1. The bold part of paths are the difference w.r.t the previous one. The guards g_0, g_1 and g_2 are as follows: $g_0 \stackrel{def}{=} x \leqslant 2 \wedge x \leqslant 1 \wedge \overleftarrow{x \leqslant 2}^{x}{}^{\varnothing}$,
$g_1' \stackrel{def}{=} x \leqslant 2 \wedge \overleftarrow{x \leqslant 1 \wedge \overleftarrow{x \leqslant 3}^{x}}^{x}$,

$g_1 \stackrel{def}{=} x \leqslant 2 \wedge x \leqslant 1 \wedge x \leqslant 3 \wedge x \leqslant 2 \wedge x \leqslant 1 \wedge x \leqslant 2 \wedge \overleftarrow{g_1'}^{\varnothing}{}^{x}{}^{\varnothing}{}^{x}{}^{\varnothing}{}^{x}{}^{\varnothing}{}^{x}{}^{\varnothing}{}^{x}{}^{\varnothing}$,

$g_{2'} \stackrel{def}{=} x \leqslant 2 \wedge x \leqslant 1 \wedge x \leqslant 2 \wedge x \leqslant 2 \wedge x \leqslant 1 \wedge x \leqslant 3 \wedge x \leqslant 2 \wedge \overleftarrow{g_1'}^{\varnothing}{}^{x}{}^{\varnothing}{}^{x}{}^{\varnothing}{}^{x}{}^{\varnothing}{}^{x}{}^{\varnothing}{}^{x}{}^{\varnothing}$,

and $g_2 \stackrel{def}{=} x \leqslant 2 \wedge x \leqslant 1 \wedge x \leqslant 3 \wedge x \leqslant 2 \wedge x \leqslant 1 \wedge x \leqslant 3 \wedge \overleftarrow{g_2}'$.

V. Test Generation from *TPAIO*

We present in this section our method for test generation from a given *TPAIO*. We first present the test generation process and then the two new steps of our method. The other step is the computation of a *RTA* that is presented in the previous section.

A. Process

The data flow diagram in Fig. 4 shows the three steps of the test generation process that we propose in this paper:

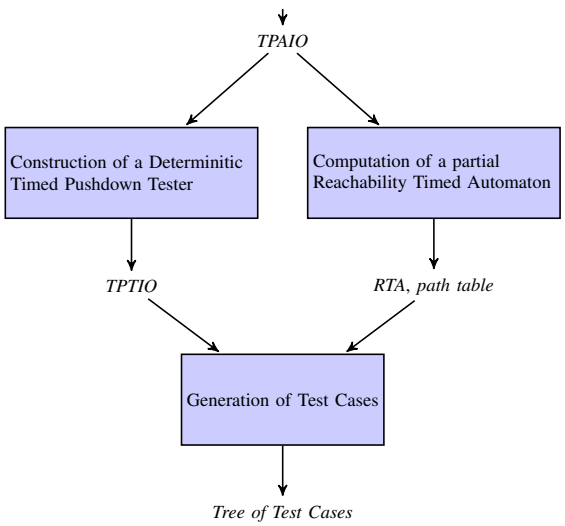

Figure 4. Test Generation from *TPAIO* Process

1) Construction of a *TPTIO* from a *TPAIO*: A *TPAIO* specifies clock constraints. For this reason, we propose to compute a Deterministic Timed Pushdown Tester with Inputs and Outputs (*TPTIO*). The tester obtained is a *TPAIO* provided with a location $fail$.
2) Computation of a partial *RTA* and its path table from the *TPAIO* presented in Sec. IV-B: popping actions depend on the content of the stack as it is impossible to pop a symbol if it is not on the top of the stack. This step computes one or many paths between two locations of *TPAIO* by respecting the stack constraints. The *RTA* is a finite timed automaton. The path table associates each transition of the *RTA* with one or many paths of the *TPAIO*. This step is presented in the previous section.
3) Generation of tree of test cases that are correct behaviours of the *TPAIO*, computed by using the *TPTIO*, the *RTA* and its path table. It is divided into two steps:(a) Generation of a tree of Test Cases (TCs) that are a tree of paths of ϵ-transitions that go from an initial to a final location of *RTA*. (b) Generation of tree of test cases of the *TPTIO*. The second step adds the location $fail$ and the transitions that leads to it.

B. Construction of a TPTIO from a TPAIO

Similarly to [2] and in order to be able to pronounce non-conformances between an IUT and the *TPAIO*, we compute a *tester* from a *TPAIO*. The tester is a *TPAIO*. Its output actions are the output actions of Σ_{out}, the stack actions of Γ^{+-} and the response delays. The tester is obtained from the *TPAIO* by enriching it with a special *fail* location and transitions that lead to it from each location l. Let Δ_l be the set of transitions leaving l in the *TPAIO*.

Definition 7 (Deterministic Timed Pushdown Tester): The *TPTIO* $T^T = (L \cup \{fail\}, l_0, \Sigma, \Gamma, \{y\}, \Delta \cup \Delta_{fail}, F)$ of a *TPAIO* $T = \langle L, l_0, \Sigma, \Gamma, X, \Delta, F \rangle$. The transitions of Δ_{fail} are computed as follows:

- (i) Let $\overline{\Delta_l}$ be the set of the complement in Γ^{+-} of the stack actions of Δ_l and the complement in Σ_{out} of the output actions of Δ_l. For all $a \in \overline{\Delta_l}$, the transition $(l, a, true, \varnothing, fail)$ is in Δ_{fail}.
- (ii) Observations, earlier or later than specified, of the stack and output actions of Δ_l: for every transition (l, a, g, X', l') in Δ_l, the transition $(l, a, \neg g, \varnothing, fail)$ is in Δ_{fail}.
- (iii) The transition $(l, -, g, \varnothing, fail)$ is in Δ_{fail} where g is the conjunction of the exceeding of the deadlines of the stack and output actions of Δ_l.

Figure 5.(a) illustrates this by showing the tester associated to the *TPAIO* of Fig 1. The label $a_1|a_2|...|a_n$ denotes the set of labels $\{a_1, a_2, ..., a_n\}$. The notations $F0$ to $F4$ are the following abbreviations: $F0 \stackrel{def}{=} ?B|?C|?E|?G|pow^+$, $F1 \stackrel{def}{=} F0|pow^-$, $F2 \stackrel{def}{=} ?E|?G|pow^+|pow^-$, $F3 \stackrel{def}{=} ?B|?C|?E|?G|pow^-$ and $F4 \stackrel{def}{=} ?B|?C|pow^+|pow^-$.

C. Generating Correct Behaviour Test Cases

Definition 8 (Tree of Test Cases): Let $T = \langle L, l_0, \Sigma, \Gamma, X, \Delta, F \rangle$ be a *TPAIO* that is a specification. A tree of test cases is a deterministic acyclic *TPAIO* whose locations are either location of T or $pass$ or $fail$.

We define firstly what a tree of test cases is in Def. 8. The usual approach described in [2] to derive tests from a tester consists of enumerating its executions and emitting the verdict *pass* when the executions don't end in the *fail* location. This often can only be done partially in practice due to the very large, if not infinite, number of possible executions. So practically a targeted set of executions are extracted out of the tester. We propose to select the executions that reach a final location with an empty stack, for producing a set of nominal test cases. For this, we select the ϵ-transitions going from an initial location to a final one. The guard of an ϵ-transition with backward closure is expressed as a system of linear inequalities over real numbers. For example, in Table II, the guard of the transition $l_0 \xrightarrow{\epsilon, g_0} l_3$ is expressed by the formula $\exists(\delta_1, \delta_2, \delta_3) \in \mathbb{R}^3$. $\delta_1 \leqslant 2 \wedge \delta_1 + \delta_2 \leqslant 1 \wedge \delta_3 \leqslant 2$. The satisfiability of a guard can be efficiently evaluated by means of SMT solvers integrating simplex based methods (see [14] for example), such as Z3 [15]. The result is a tree of test cases, in which the actions are either observable (the stack actions of Γ^{+-} and the output actions of Σ_{out}) or controllable (the input action of Σ_{in}). The leaves of the tree other than *fail* are replaced by the verdict *pass*. Figure 5.(b) shows a tree of test cases that present seven paths

Transitions	Paths
$(l_0, \epsilon, g_0, \varnothing, l_3)$	$l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{B,x\leqslant 1,\{x\}} l_2 \xrightarrow{D,x\leqslant 2} l_3$
$(l_0, \epsilon, g_1, \varnothing, l_7)$	$l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{B,x\leqslant 1,\{x\}} l_2 \xrightarrow{D,x\leqslant 2} l_3 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{E,x\leqslant 1,\{x\}} l_6 \xrightarrow{F,x\leqslant 3} l_7$
$(l_0, \epsilon, g_1, \varnothing, l_9)$	$l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{B,x\leqslant 1,\{x\}} l_2 \xrightarrow{D,x\leqslant 2} l_3 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{\mathbf{G,x\leqslant 1,\{x\}}} l_8 \xrightarrow{\mathbf{H,x\leqslant 3}} l_9$
$(l_0, \epsilon, g_2, \varnothing, l_7)$	$l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{B,x\leqslant 1,\{x\}} l_2 \xrightarrow{D,x\leqslant 2} l_3 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{G,x\leqslant 1,\{x\}} l_8 \xrightarrow{H,x\leqslant 3} l_9 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{E,x\leqslant 1,\{x\}} l_6 \xrightarrow{F,x\leqslant 3} l_7$
	$l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{B,x\leqslant 1,\{x\}} l_2 \xrightarrow{D,x\leqslant 2} l_3 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{\mathbf{E,x\leqslant 1,\{x\}}} l_6 \xrightarrow{\mathbf{F,x\leqslant 3,\{x\}}} l_7 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{E,x\leqslant 1,\{x\}} l_6 \xrightarrow{F,x\leqslant 3} l_7$
$(l_0, \epsilon, g_2, \varnothing, l_9)$	$l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{B,x\leqslant 1,\{x\}} l_2 \xrightarrow{D,x\leqslant 2} l_3 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{G,x\leqslant 1,\{x\}} l_8 \xrightarrow{H,x\leqslant 3,\{x\}} l_9 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{G,x\leqslant 1,\{x\}} l_8 \xrightarrow{H,x\leqslant 3} l_9$
	$l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{C,x\leqslant 1,\{x\}} l_4 \xrightarrow{pow^+,x\leqslant 3,\{x\}} l_0 \xrightarrow{A,x\leqslant 2} l_1 \xrightarrow{B,x\leqslant 1,\{x\}} l_2 \xrightarrow{D,x\leqslant 2} l_3 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{\mathbf{E,x\leqslant 1,\{x\}}} l_6 \xrightarrow{\mathbf{F,x\leqslant 3,\{x\}}} l_7 \xrightarrow{pow^-,x\leqslant 2,\{x\}} l_5 \xrightarrow{G,x\leqslant 1,\{x\}} l_8 \xrightarrow{H,x\leqslant 3} l_9$

TABLE II. Example of transitions of the *RTA* of the *TPA* of Fig. 1, with their paths

corresponding to the following ϵ-transitions: $(l_0, g_0, \varnothing, l_3)$, $(l_0, g_1, \varnothing, l_7)$, $(l_0, g_1, \varnothing, l_9)$, $(l_0, g_2, \varnothing, l_7)$ and $(l_0, g_2, \varnothing, l_9)$ whose the paths can be seen in the Table II.

VI. Soundness, Incompleteness and Coverage of Method

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

A. Soundness

Definition 9 (reachability in an RTA): A location l_i is reachable in an *RTA* iff there exists a run that leads to it from the initial location l_0.

To prove the reachability of a location l_i in an *RTA*, we compute a sequence of ϵ-transitions that lead from l_0 to l_i where all the clock constraints are satisfied. When the sequence is a single ϵ-transition, it is sufficient to evaluate the satisfiability of its guard. In a longer sequence, the clock constraints are composed by means of backward closures but not verified, whereas the constraints on the stack are already verified by construction with the rules RA_1 to RA_4. Consequently we can get rid of the stack constraints, and see the corresponding transitions as ϵ-ones. By merging all these successive ϵ-transitions by the rule RA_4, we finally get only one ϵ-transition that leads from l_0 to l_i. Deciding of the reachability of l_i thus reduces to evaluate the satisfiability of the guard of this ϵ-transition.

Definition 10 (reachability in a TPA): A location l_i is reachable in a *TPA* iff there exists a run that leads to the location l_i from the initial location l_0.

Theorem 1: A location l is reachable in a *TPA* iff it is reachable in its *RTA*.

Proof: This is a direct consequence of Lemma 2. ■

Proposition 1: Let $\pi = l_0 \xrightarrow{a_0,g_0,X_0} l_1 \xrightarrow{a_1,g_1,X_1} l_2 \xrightarrow{a_2,g_2,X_2} ...l_{n-1} \xrightarrow{a_{n-1},g_{n-1},X_{n-1}} l_n \xrightarrow{a_n,g_n,X_n} fail$ be a path of a tree of test cases of a specification $T = \langle L, l_0, \Sigma_{in} \cup \Sigma_{out}, \Gamma, X, \Delta, F\rangle$ where $l_i \in L$, $g_i \in Grd(X)$ and $a_i \in \Sigma_{in} \cup \Sigma_{out} \cup \Gamma^{+-} \cup \{-\}$ for each $0 \leqslant i \leqslant n$. If a verdict $fail$ is observed while executing π on the implementation I, then the implementation I is 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 a trace of a run of the path π. $(l_n, v_n + \delta_n, p_n)$ is the current state after the execution of $\delta_0 a_0 \delta_1 a_1 \delta_2 ... \delta_{n-1} a_{n-1} \delta_n$. It exists the three following cases to reach $fail$:

- fail is detected after having observed a_n in the case of not acceptable stack or output action by the specification according to item (i) in Def. 7 of the tester. If a_n is in the complement of the stack actions of Δ_{l_n} w.r.t Γ^{+-} or a_n is in the complement of the output actions of Δ_{l_n} w.r.t Σ_{out}, then, this transition $(l_n, a_n, true, \varnothing, fail)$ is a transition of the tester. Therefore, $a_n \notin out(T\ after\ \delta_0 a_0 \delta_1 a_1 \delta_2 ... \delta_{n-1} a_{n-1} \delta_n)$ and I does not conform to T.
- fail is detected after having observed a_n in the case of the observations of, earlier or later stack action or output actions by the specification according to the item (ii) in Def. 7 of the tester. a_n not exist in the complement of the stack actions of Δ_{l_n} w.r.t Γ^{+-} or in the complement of the output actions of Δ_{l_n} w.r.t Σ_{out}. The current clock valuation does not satisfy the guard g_n. Thus, $a_n \notin out(T\ after\ \delta_0 a_0 \delta_1 a_1 \delta_2 ... \delta_{n-1} a_{n-1} \delta_n)$ and I does not conform to T.
- fail is detected after having observed a delay δ_n in the case according to item (iii) in Def. 7 of the tester. g_n is the conjunction exceeding of the deadlines of stack and output actions of Δ_{l_n} by the specification. The transition $(l_n, -, g_n, \varnothing, fail)$ is a transition of the tester. If $v_n + \delta_n \models g_n$, then $\delta_n \notin out(T\ after\ \delta_0 a_0 \delta_1 a_1 \delta_2 ... \delta_{n-1} a_{n-1})$ and I does not conform to T.

■

For every non-conformance detected by a path of a tree of test cases there is a non conformance between the implemen-

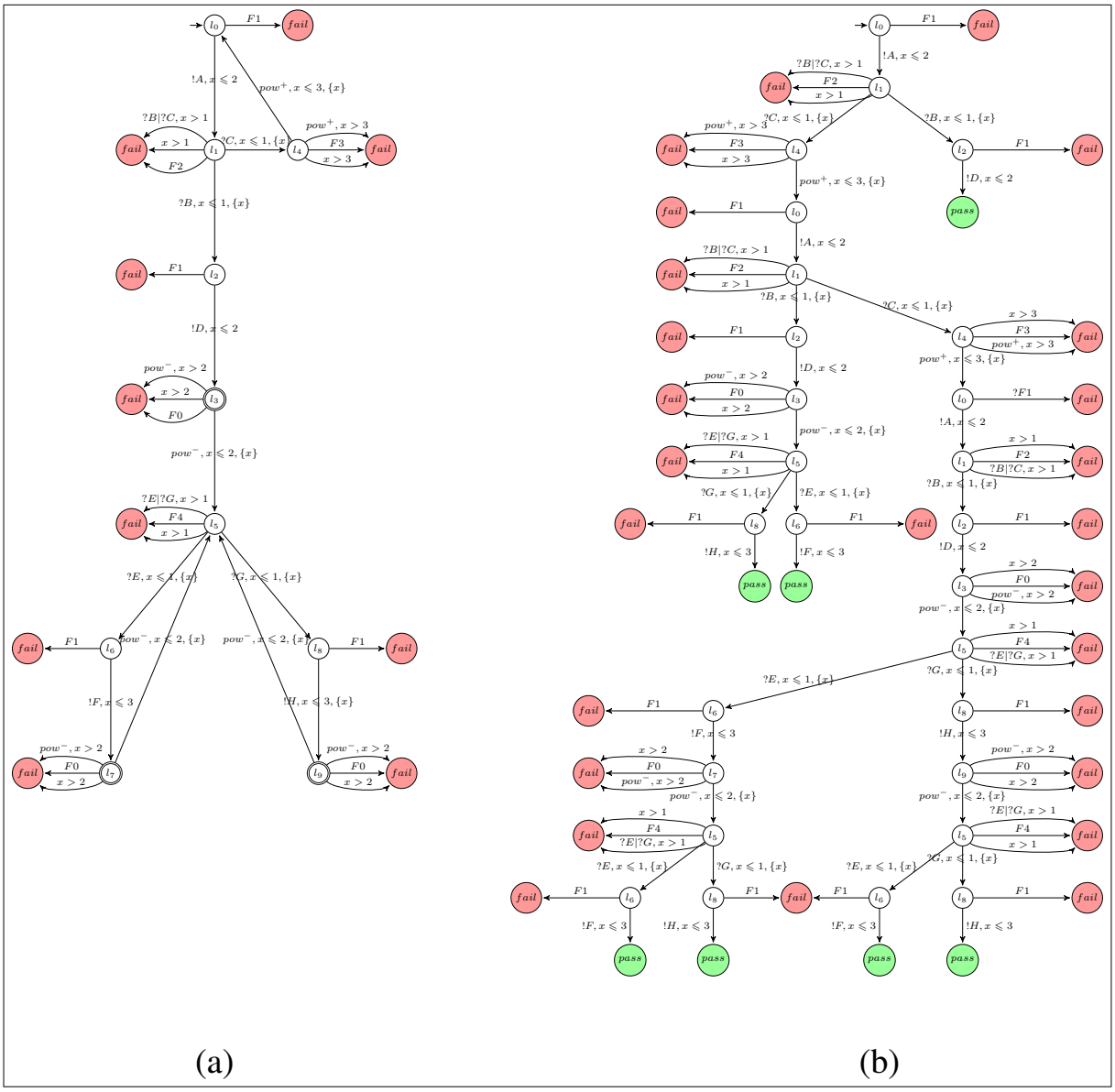

Figure 5. (a). The tester associated to the *TPAIO* of Fig 1 and (b) A tree of test cases

tation and the specification (*TPAIO*).

B. Incompleteness

The polynomial complexity of the algorithm of Fig. 3 is possible thanks to the incompleteness. In the case of a *PA*, there is an ϵ-transition in the *RTA* between two locations l and l' if and only if l' is reachable from l in the *PA* [5]. This is w.r.t. the stack constraints. But in the case of a *TPA*, the reachability also depends on the clock constraints: it is sufficient that the guard of an ϵ-transition from l to l' is satisfiable for l' to be reachable from l in the *TPA*, but it is not necessary. If the guard is not satisfiable, l' might still be reachable from l, but through another path. As we have dropped some of the possible such alternative paths, we cannot conclude anymore that the location is completely unreachable.

Figure 6 illustrates the incompleteness of our algorithm. Figure 6(a) presents a *TPA*. Figure 6(b) shows the *RTA* obtained by our algorithm. The location l_1 is not reachable in this *RTA*, because the two guards of the transitions $l_0 \xrightarrow{\epsilon,\ldots} l_1$ are not satisfiable. However, l_1 is reachable in the *TPA*. It could have been detected if the transition

$$l_0 \xrightarrow{\epsilon, x\leqslant 1\wedge x\leqslant 1\wedge x\leqslant 1\wedge x=0\wedge y\geqslant 3\wedge x\leqslant 2\wedge x\leqslant 2, \{x\}} l_1$$

had been added by applying the rule RA_3. It has not been added because it covers no a new transition between l_0 and l_1.

C. Coverage

In section IV-B, we have presented a method for computing an *RTA* from a *TPAIO*. The algorithm that computes the *RTA* takes into account the coverage of the transitions of the *TPAIO*. It adds a new ϵ-transition (l, ϵ, g, X, l') only if its path covers a new transition of the *TPAIO* between the locations l and l'. The paths of all the timed ϵ-transitions that go to a final

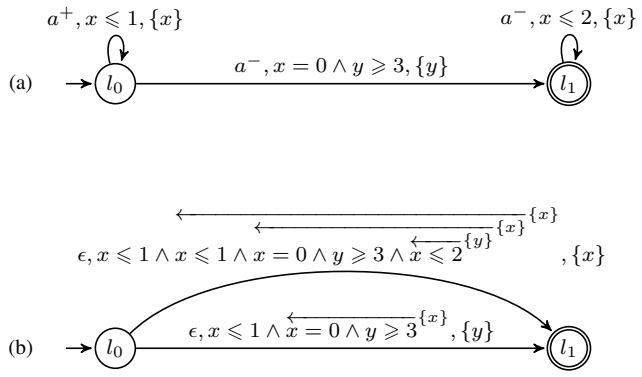

Figure 6. Example of an *RTA* (b) incompletely catching the reachability of a *TPA* (a)

location of the *RTA* cover all the transitions of the *TPAIO*. But we can not conclude that the indeed test cases cover all the transitions of *TPAIO*, because the guard of a given timed ϵ-transition may not be satisfiable. But, if the guards of all the ϵ-transitions that go from an initial location to final location are satisfiable, then all the reachable locations and transitions of the *TPAIO* are covered. This is the case in our example.

VII. Conclusion and Further Work

We presented a method to generate test from *TPAIO* that to our knowledge has not been treated in the literature. First, we presented a method that adapts the algorithm defined in [5] for computing an *RTA* from a *TPA*. The clock constraints are treated by means of backward closure computations, which gives a system of linear inequalities whose satisfiability evaluation is entrusted to an SMT-solver. To keep the method polynomial the computed *RTA* is incomplete, but it is sound. We additionally compute a path table which associates each transition of the *RTA* to its paths that can be unfolded as a sequence of transitions of the *TPA*. Second, this path table allows the generation of tests that cover the locations and transitions found reachable of the *TPA*. Third, we have distinguished between the inputs and outputs of the *TPA* for checking the conformance adapted from *TAIO* to *TPAIO* of the IUT w.r.t its specification. A further work is to modify the rules defining the *RTA* for dealing with the ϵ-transition cycles while defining a finite *RTA*. This will lead to a complete, but exponential, algorithm by saturation, that could experimentally be compared for completeness with the one of this paper.

References

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

[2] J. Tretmans, "Test generation with inputs, outputs and repetitive quiescence," Software - Concepts and Tools, vol. 17, no. 3, 1996, pp. 103–120.

[3] P.-C. Héam and C. Masson, "A random testing approach using pushdown automata," in Tests and Proofs, ser. LNCS, 2011, vol. 6706, pp. 119–133.

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

[5] A. Finkel, B. Willems, and P. Wolper, "A direct symbolic approach to model checking pushdown systems (ext. abs.)," in Infinity97, ser. ENTCS, vol. 9, 1997, pp. 27–37.

[6] A. Bouajjani, J. Esparza, and O. Maler, "Reachability analysis of pushdown automata: Application to model-checking," in CONCUR, ser. LNCS, vol. 1243, 1997, pp. 135–150.

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

[8] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-checking in dense real-time," Inf. Comput., vol. 104, no. 1, 1993, pp. 2–34.

[9] 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.

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

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

[12] B. Bérard, A. Petit, V. Diekert, and P. Gastin, "Characterization of the expressive power of silent transitions in timed automata," Fundamenta Informaticae, vol. 36, no. 2-3, 1998, pp. 145–182.

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

[14] B. Dutertre and L. de Moura, "A fast linear-arithmetic solver for DPLL(T)," in CAV, ser. LNCS, vol. 4144, 2006, pp. 81–94.

[15] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in TACAS, ser. LNCS, vol. 4963, 2008, pp. 337–340.