# Conformance Testing for Timed Recursive Programs

Hana M'Hemdi, Jacques Julliand, Pierre-Alain Masson, and Riadh Robbana

**Abstract** This paper is about conformance testing of timed pushdown automata with inputs and outputs (TPAIO), that specify both stack and clock constraints. TPAIO are used as a model for timed recursive programs. This paper proposes a novel method of off-line test generation from deterministic TPAIO. In this context, a first problem is to resolve the clock constraints. It is solved by computing a deterministic timed pushdown tester with inputs and outputs (TPTIO), that is a TPAIO with only one clock, and provided with a location fail. To generate test cases from a TPTIO, we compute from it a finite reachability automaton (RA), that relates any of its transitions to a path of the TPTIO. The RA computation takes the TPTIO transitions as a coverage criterion. The transitions of the RA, thus the paths of the TPTIO are used for generating test cases that aim at covering the reachable locations and transitions of the TPAIO.

**Key words:** Timed Automata; Timed Pushdown Automata; Conformance testing; Approximated determinization; Analog-clock testing; Reachability automaton.

## 1 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

H. M'Hemdi · J. Julliand · P.-A. Masson
FEMTO-ST/DISC, University of Franche-Comté, 16, route de Gray 25030 Besançon, France
e-mail: {hana.mhemdi, jacques.julliand, pierre-alain.masson}@femto-st.fr

H. M'Hemdi
LIP2, University of Tunis El Manar, Tunisia

R. Robbana
LIP2 and INSAT-University of Carthage, Tunisia
e-mail: riadh.robbana@fst.rnu.tn

generation, that are very active research areas, rely on exploiting these models. This paper is about generating tests from the model of 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*, by including both a stack and some clocks, can model recursive systems with inputs and outputs whose execution is in a real time context.

Test generation from *TPAIO* could apply to industrial case studies such as that of [6], 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 [16], this system can be modelled as a recursive timed automaton with a safety critical objective. [4] argues that timed recursive state machines are related to an extension of pushdown timed automata, where an additional stack, coupled with the standard control stack, is used to store temporal valuations of clocks. Therefore, the system of [6] 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 [4].

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 recursive deterministic programs without inputs.

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 [5][1], and time exponential [7].

Our first contribution is the construction of a timed pushdown tester with inputs and outputs (*TPTIO*) of the *TPAIO*, by adapting the determinization method of [11] (for timed IO automata with no stack) to the *TPAIO* case. Even in the restricted framework of deterministic programs, this step is useful as it produces a model with one single clock reset after each transition, in which locations reachability has been verified w.r.t. the satisfiability of the clock constraints. Additionally, this will facilitate a future extension of the method to the case of non-deterministic *TPAIO*. Fail verdicts are added as special locations to this determinized *TPAIO*: 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 [9] 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* whose transitions are all related to one path of the *TPTIO*. 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 set of test cases from a *TPAIO*.

To summarize, our contributions are to: $(i)$ define *tpioco*: a conformance relation for the *TPAIO* model; $(ii)$ adapt the determinization method of [11] to obtain a *TPTIO*; $(iii)$ compute an *RA* where any transition is labelled with a path of a *TPTIO*, by adapting the reachability computation of [9]; $(iv)$ generate test cases by covering the reachable locations and transitions of the *TPAIO*. To our knowledge these problems, solved for the TA [11] and PA [9], have not been handled for the *TPAIO* yet.

The paper is organized as follows. Section 2 presents the *TA* model and the timed input-output conformance relation *tioco*. Our model of *TPAIO* is presented in Sect. 3, together with a *TPAIO* conformance relation. Our *TPAIO* test generation method is presented in Sect. 4. We discuss the soundness, incompleteness and coverage of our method in Sect. 5. We conclude and indicate future works in Sect. 6.

## 2 Background

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

### *2.1 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, $\Sigma$ 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 $\Delta^T$ is the set of transitions. There are two kinds of transitions in $\Delta^T$: 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 $\delta$. 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 $\pi$ 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. $\sigma = (l_0,v_0) \rightarrow^{\delta_0} (l_0,v_0+\delta_0) \rightarrow^{a_0} (l_1,v_1) \rightarrow^{\delta_1} (l_1,v_1+\delta_1) \rightarrow^{a_1} (l_2,v_2) \rightarrow^{\delta_2} \ldots \rightarrow^{a_{n-1}} (l_n,v_n)$ where $\delta_i \in \mathbb{R}^+$ and $a_i \in \Sigma$ for each $0 \leqslant i \leqslant n-1$ is a run of $\pi$ if $v_i \models g_i$ for $0 \leqslant i < n$. A run alternates timed and discrete transitions. 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 $\Sigma$. $P_{\Sigma_1}(\rho)$ is the projection on $\Sigma_1 \subseteq \Sigma$ of a trace $\rho$ 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 $\rho$. For example, $Time(5a42) = 11$. $s_0^T \rightarrow^\rho s$ means that the state $s$ is reachable from the initial state $s_0^T$, i.e. there exists a run $\sigma$ from $s_0^T$ to $s$ whose trace is $\rho$. $s_0^T \rightarrow^\rho$ means that there exists $s'$ such that $s_0^T \rightarrow^\rho s'$.

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 $\Sigma_{in}$ is a set of input actions, $\Sigma_{out}$ is a set of output actions and $\tau$ 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 $\Sigma_{in}$ and observes outputs of $\Sigma_{out}$. The implementation under test ($IUT$), sends observable actions of $\Sigma_{out}$ and accepts commands of $\Sigma_{in}$.

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 $\Sigma_\tau$ 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 $\Delta$ then $g_1 \wedge g_2$ is not satisfiable. It is observable if no transition is labelled by $\tau$. $Reach(T) = \{s^T \in S^T \mid \exists \rho.(\rho \in RT(\Sigma) \wedge s_0^T \rightarrow^\rho s^T\}$ denotes the set of reachable states of a *TAIO* $T$. A *TAIO* $T$ is non blocking if $\forall (s,\delta).(s \in Reach(T) \wedge \delta \in \mathbb{R}^+ \Rightarrow \exists \rho.(\rho \in RT(\Sigma_{out} \cup \{\tau\}) \wedge Time(\rho) = \delta \wedge s \rightarrow^\rho))$. A *TAIO* is called input-complete if it accepts any input at any state.

## 2.2 *Timed Input-Output Conformance Relation* $tioco$

We first present the conformance theory for timed automata based on the conformance relation *tioco* [11]. *tioco* is an extension of the *ioco* relation of Tretmans [15]. The main difference is that *ioco* uses the notion of quiescence, contrarily to *tioco* in [11] 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* $\rho = \{s \in S^T \mid \exists \rho'.(\rho' \in RT(\Sigma_\tau) \wedge s_0^T \rightarrow^{\rho'} s \wedge P_\Sigma(\rho') = \rho)\}$ is the set of states of $T$ that can be reached by a trace $\rho'$ whose projection $P_\Sigma(\rho')$ on the controllable and observable actions is $\rho$.
- $ObsTTraces(T) = \{P_\Sigma(\rho) \mid \rho \in RT(\Sigma_\tau) \wedge s_0^T \rightarrow^\rho\}$ is the set of observable timed traces of a *TAIO* $T$.

- $elapse(s) = \{\delta \mid \delta > 0 \wedge \exists \rho.(\rho \in RT(\{\tau\}) \wedge Time(\rho) = \delta \wedge s \rightarrow^{\rho})\}$ is the set of delays that can elapse from the state $s$ with no observable action.
- $out(s) = \{a \in \Sigma_{out} \mid s \rightarrow^{a}\} \cup elapse(s)$ 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 $T$.

## 3 Model and Conformance Relation

This section defines our model: *TPAIO*, as well as *tpioco*, a conformance relation for *TPAIO*. We show an example of a *TPAIO* that models a recursive program.

### *3.1 Timed Pushdown Automata with Inputs and Outputs*

A *TPAIO* $T = \langle L, l_0, \Sigma, \Gamma, X, \Delta, F \rangle$ is a *TAIO* 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 $(l, v, p)$ with $l$ a location of the *TPAIO*, $v$ a clock valuation in $X \rightarrow \mathbb{R}^+$ and $p$ a stack content in $\Gamma^*$.

**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 $\Sigma_{in}$ is a finite set of input actions, $\Sigma_{out}$ is a finite set of output actions and $\{\tau\}$ is an internal and unobservable action, $\Gamma$ 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 $\Gamma^{+-}$ 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 and pop operations. Any *TPAIO* can be normalized since any

*PA* can be normalized [14]. Due to the class of application, we consider in the remainder of the paper that the *TPAIO* are normalized deterministic timed pushdown automata with outputs and without inputs. We denote $a$ the actions of $\Gamma$ and $A$ the actions of $\Sigma_{out}$.

We define *tpioco*, our *TPAIO* conformance relation, as an extension of the *tioco* conformance relation [11]. It is the same relation as *tioco* for *TAIO* by considering the whole alphabet to be $\Sigma_{out} \cup \Gamma^{+-} \cup \{\tau\}$ instead of $\Sigma_{in} \cup \Sigma_{out} \cup \{\tau\}$. The output alphabet is $\Sigma_{out} \cup \Gamma^{+-}$ instead of $\Sigma_{out}$ and there is no input alphabet.

## *3.2 Modelling of Recursive Programs*

Figure 1 shows a program that recursively computes the $n^{th}$ Fibonacci number, with instructions labels from $l_0$ to $l_6$. We abstract the control flow graph of a recursive program by a *PA*, as explained in [8]. Figure 2 shows a *TPAIO* that abstracts the program of Fig. 1. Here the time constraints have been added arbitrarily for illustration purposes. The location labels are the instruction labels in the program of Fig. 1. $Fib_1^+$ and $Fib_2^+$ are respectively the (push) calls $Fib(n-1)$ and $Fib(n-2)$. $Fib_1^-$ and $Fib_2^-$ are respectively the (pop) returns from the calls $Fib(n-1)$ and $Fib(n-2)$. Thus $\Gamma = \{Fib_1, Fib_2\}$. Such an example is a transformational system in which the tester observes any action of the program. Therefore, we choose that all the executions of atomic instructions and conditions are in $\Sigma_{out}$. They are labelled from $A$ to $E$ as follows: $A \stackrel{def}{=}$ int $res_1$, $res_2$, $B \stackrel{def}{=} n \leqslant 1$, $C \stackrel{def}{=} n > 1$, $D \stackrel{def}{=}$ *return* $n$ and $E \stackrel{def}{=}$ *return* $res_1 + res_2$. We use the notation $!act$ to denote the output action $act$.

```
int  Fib(int n)
    l0 : int res1, res2;
    if  l1 : n ⩽ 1 then
        l2 : return n;
    else
        l3 : res1 = Fib(n−1); //Fib1+
        l4 : res2 = Fib(n−2); //Fib2+
        l5 : return res1 + res2;
    fi
l6 : end.
```

Fig. 1. A Fibonacci Computation Program $Fib(n)$

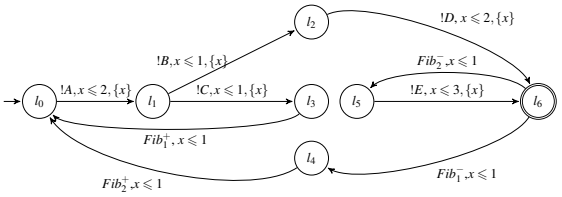


Fig. 2. A *TPAIO* Modelling the Program of Fig. 1

# 4 Test Generation from *TPAIO*

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

## *4.1 Test Generation Process*

The data flow diagram of Fig. 3 shows the three steps of the test generation process that we propose in this paper:

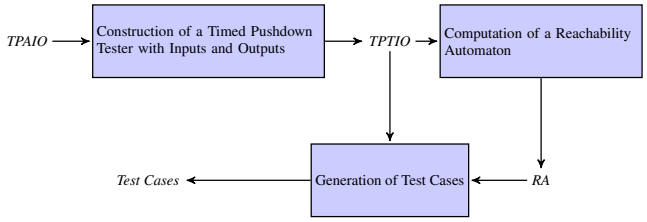


Fig. 3. Test Generation Process from a *TPAIO*

1. Construction of a *TPTIO* of a *TPAIO*: A *TPAIO* specifies clock constraints. For this reason, we propose to compute a Timed Pushdown Tester with Inputs and Outputs (*TPTIO*) that resolves the clock constraints. The tester obtained is a *TPAIO* with one clock, reset each time the tester observes an action, and provided with a location $fail$.
2. Computation of an *RA* from the *TPTIO*: pop actions depend on the content of the stack. This step computes one or many paths between two symbolic locations of a *TPTIO* by respecting the stack constraints, i. e. such that the stack is empty in the target location. The *RA* is a finite automaton with any of its transition related to one path of the *TPTIO*. Such a transition is called a $\pi$-transition.
3. Generation of test cases as correct behaviours of the *TPAIO*, computed from the *TPTIO*. There are two sub-steps: (a) generation of test paths of $\pi$-transitions that go from an initial to a final location of the *RA*; (b) generation of *TPTIO* test cases, by adding to the test paths the location $fail$ and the transitions that lead to it.

### *4.2 Construction of a TPTIO from a TPAIO*

In [11], Krichen and Tripakis propose a method for conformance testing of non-deterministic *TAIO*. They propose an algorithm for generating test cases. They compute a tester that has only one clock, reset each time the tester observes an action.

We propose to adapt the method of [11] for computing a *TPTIO* from a *TPAIO*. Let $out(l)$ be the set of transitions leaving the location $l$ in the *TPAIO*. A verdict $fail$ is emitted if either an unspecified stack or output action is observed, or a stack or output actions of $out(l)$ is observed earlier or later than specified, or a timeout occurs. A *TPTIO* has only one clock which is reset every time the tester observes an action. As a consequence, all the guards of a *TPTIO* are satisfiable. We define a *TPTIO* of a *TPAIO* in Def. 4.

**Definition 4 (*TPTIO*).** The *TPTIO* $T^T = (L^T, l_0^T, \Sigma_{out}, \Gamma, \{y\}, \Delta^T, F^T)$ of a *TPAIO* $T = \langle L, l_0, \Sigma_{out} \cup \{\tau\}, \Gamma, X, \Delta, F\rangle$ is a *TPAIO* with only one clock $y$ that is a new clock w.r.t $X$ where:

- $L^T \subseteq (L \times CC(X \cup \{y\})) \cup \{fail\})$ 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,
- $\Delta^T \subseteq L^T \times \Sigma_{out} \cup \Gamma^{+-} \times Grd(\{y\}) \times \{y\} \times L^T$ is a finite set of transitions.

The computation, taken from [11], of a partition where each part is in $Grd(\{y\})$ is as follows: let $K$ be the greatest constant appearing in a constraint of a given symbolic location $l^T$ or in a guard of a given transition of $T$. The following set of intervals is a partition: $\{[0,0],]0,1[,[1,1],]1,2[,...,[K,K],\ ]K,\infty[\}$. Before presenting the method to compute $\Delta^T$, we need to define the following sets:

- $\Delta_a = \{t \mid t \in \Delta \wedge \exists(l,g,X',l').(t = (l,a,g,X',l'))\}$ is the set of transitions of $\Delta$ labelled by $a$.
- $\Delta_a((l,v),u) = \{(l,a,g,X',l') \in \Delta_a \mid v \wedge u \wedge g \text{ satisfiable}\}$ is the set of transitions labelled by $a$ whose guards are satisfied by $(l,v)$ where $v$ is in $CC(X \cup \{y\})$ and the clock $y$ is equal to $u$.

Since our model is that of deterministic *TPAIO*, $\Delta_a$ and $\Delta_a(l^T,u)$ contain at most one transition. For all intervals $u$, the coarsest partition is obtained from $l^T$ by taking the union of the intervals that have the same set $\Delta_a(l^T,u)$. For a symbolic location $l^T \in L^T$ of $T^T$, $a \in \Sigma_{out} \cup \Gamma^{+-}$:

- $usucc(l^T) = l^{T'}$ such that $\exists\rho.(\rho \in RT(\{\tau\}) \wedge l^T \rightarrow^{\rho} l^{T'})$ is the symbolic location reachable from $l^T$ by applying a sequence of unobservable actions.
- $dsucc(l^T,a) = l^{T'}$ such that $l^T \rightarrow^{a} l^{T'}$ is the symbolic location reachable from $l^T$ by applying the action $a$.

In the initial location $l_0^T$, all the clocks equal zero, including $y$. The construction of the *TPTIO* repeats the following step: selection of a symbolic location $l^T \in L^T$ and application of the following possibilities to add new transitions to $\Delta^T$: ***(i)*** output and stack actions: for every action $a \in \Sigma_{out} \cup \Gamma^{+-}$, for every coarsest partition $u$,

if $\Delta_a(l^T,u) = \varnothing$ then the transition $(l^T,a,u,\{y\},fail)$ is added to $\Delta^T$. Otherwise the transition $(l^T,a,u,\{y\},usucc(dsucc(l^T \cap u,a)))$ is added to $\Delta^T$; ***(ii)*** timeout: let $K$ be the greatest constant appearing in the constraint of $l^T$ or in a guard of the transitions leaving $l^T$. Then, the transition $(l^T,-,y>K,\{y\},fail)$ is added to $\Delta^T$.

The first symbolic location selected is $l_0^T$. Adding new transitions to a *TPTIO* implies adding new symbolic locations to the *TPTIO*. The algorithm terminates when all the new symbolic locations are selected and treated.

Notice that the number of locations of the *TPTIO* could scale exponentially with that of the *TPAIO*. However, the impact of this blowup can be limited by putting time constraints on blocks of instructions rather than on single instructions.

*Example 1.* Figure 4 shows the *TPTIO* of the *TPAIO* of Fig. 2 where $v_i \stackrel{def}{=} 0 \leqslant x-y \leq i$. The label $a_1|a_2|...|a_n$ denotes the set of labels $\{a_1,a_2,...,a_n\}$. $F = Fib_1^+|Fib_1^-|Fib_2^+|Fib_2^-|?A|?B|?C|?D|?E$, $F0 = F \setminus \{?A\}$, $F1 = F \setminus \{?B,?C\}$, $F2 = F \setminus \{?D\}$, $F3 = F \setminus \{Fib_1^+\}$, $F4 = F \setminus \{Fib_2^+\}$, $F5 = F \setminus \{?E\}$ and $F6 = F \setminus \{Fib_1^-,Fib_2^-\}$.

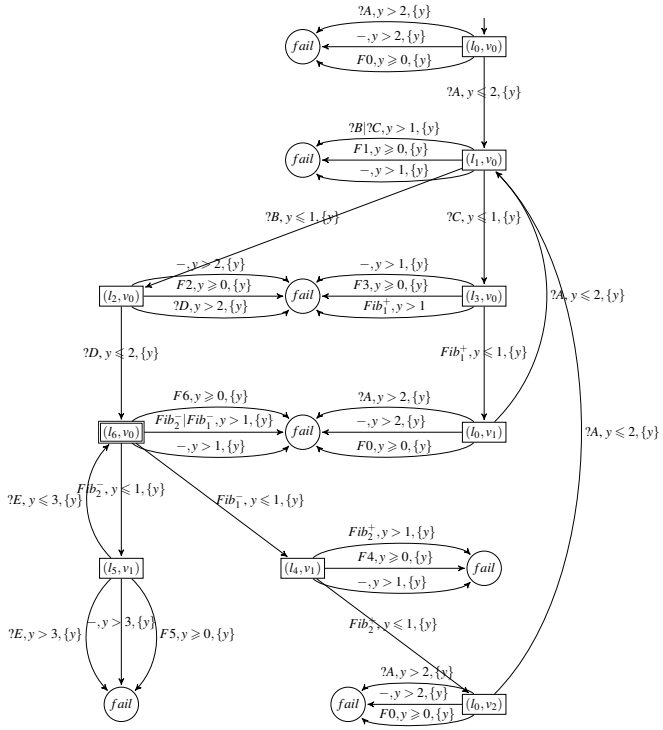


Fig. 4. The *TPTIO* of the *TPAIO* of Fig. 2

### *4.3 Reachability Automaton of a TPTIO Computation*

A *TPAIO* does not only specify clock constraints but also stack constraints. Therefore, applying the algorithm of [11] for generating analog-clock tests from *TPAIO* is not sufficient. It is necessary, for avoiding system deadlocks, to additionally take the stack content into account. For example, the pop action of a symbol that would not be on top of the stack would provoke a deadlock. We compute a reachability automaton for taking the stack constraints into account.

Let $(L^T, l_0^T, \Sigma_{out}, \Gamma, \{y\}, \Delta^T, F^T)$ be a *TPTIO*. We propose to compute a representation of its reachable locations from its initial location. This representation is called the *reachability automaton* of the *TPTIO*. It is a finite automaton whose transition labels are sequences of transitions of the *TPTIO* ($\in \Delta^{T^*}$). A $\pi$-transition $(l^T, \pi, l^{T'})$ is a transition that reaches the symbolic location $l^{T'}$ from $l^T$ and leaves the stack unchanged at the end, by taking the path $\pi$. We propose in Def. 5 the rules $R_1$ to $R_4$ that, applied repeatedly, define the *RA*.

**Definition 5 (*RA* of a *TPTIO*).** The *RA* of a *TPTIO* $(L^T, l_0^T, \Sigma_{out}, \Gamma, \{y\}, \Delta^T, F^T)$ is the automaton $(L^R, l_0^T, (\Delta^T)^*, \Delta^R, F^T)$ 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 $t_1 \stackrel{def}{=} (l_1^T, a^+, g_1, \{y\}, l_2^T)$, $t_2 \stackrel{def}{=} (l_2^T, a^-, g_2, \{y\}, l_3^T)$ and $t_3 \stackrel{def}{=} (l_3^T, a^-, g_3, \{y\}, l_4^T)$ be three transitions in $\Delta^T$ where $l_2^T$, $l_3^T$, $l_4^T$ differ from the location $fail$.

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

We have adapted an algorithm by Finkel et al. [9], that originally computes an *RA* from a *PA*, to compute an *RA* from a *TPTIO*. Our modifications are as follows:

1. we compute the path of each transition. Any $\pi$-transition in the *RA* corresponds to the path $\pi$ in the *TPTIO*;
2. the problem addressed in [9] being to check the location reachability, the paths are ignored and there is only one transition between two symbolic locations $l^T$ and $l^{T'}$. 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 $R_1$, the transitions that emit a symbol of $\Sigma_{out}$ in addition to the push and pop ones;
4. the reflexive transitions are not used in [9] because they don't cover any new state. By contrast in the rule $R_4$, we possibly extend an existing $\pi$-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 $\pi$-transition $(l^T, \pi, l^{T'})$ is performed only if $\pi$ covers a new transition between $l^T$ and $l^{T'}$ of the *TPTIO*. Thus our algorithm applies the rules $R_1$ to $R_4$ to compute the smallest set of transitions $\Delta^R$ that covers all the transitions.

*Example 2.* There are two transitions $((l_0, v_0), \pi, (l_6, v_0))$ that go from the initial symbolic location $(l_0, v_0)$ to the accepting symbolic location $(l_6, v_0)$:

1. $\pi \stackrel{def}{=} (l_0, v_0) \xrightarrow{A, y \leqslant 2, \{y\}} (l_1, v_0) \xrightarrow{B, y \leqslant 1, \{y\}} (l_2, v_0) \xrightarrow{D, y \leqslant 2, \{y\}} (l_6, v_0)$
2. $\pi \stackrel{def}{=} (l_0, v_0) \xrightarrow{A, y \leqslant 2, \{y\}} (l_1, v_0) \xrightarrow{C, y \leqslant 1, \{y\}} (l_3, v_0) \xrightarrow{Fib_1^+, y \leqslant 1, \{y\}} (l_0, v_1) \xrightarrow{A, y \leqslant 2, \{y\}} (l_1, v_0) \xrightarrow{B, y \leqslant 1, \{y\}}$
   $(l_2, v_0) \xrightarrow{D, y \leqslant 2, \{y\}} (l_6, v_0) \xrightarrow{Fib_1^-, y \leqslant 1, \{y\}} (l_4, v_1) \xrightarrow{Fib_2^+, y \leqslant 1, \{y\}} (l_0, v_2) \xrightarrow{A, y \leqslant 2, \{y\}} (l_1, v_0) \xrightarrow{B, y \leqslant 1, \{y\}}$
   $(l_2, v_0) \xrightarrow{D, y \leqslant 2, \{y\}} (l_6, v_0) \xrightarrow{Fib_2^-, y \leqslant 1, \{y\}} (l_5, v_1) \xrightarrow{E, y \leqslant 3, \{y\}} (l_6, v_0)$.

## *4.4 Generation of Correct Behaviour Test Cases*

**Definition 6 (Test Case).** Let $T = \langle L, l_0, \Sigma_{out} \cup \{\tau\}, \Gamma, X, \Delta, F \rangle$ be a *TPAIO* specification and $T^T = (L^T, l_0^T, \Sigma_{out}, \Gamma, \{y\}, \Delta^T, F^T)$ be the *TPTIO* of $T$. A test case is a deterministic acyclic *TPAIO* whose locations are either configurations $(l, v, p)$, or *pass*, or $fail$, or $stack\_fail$.

We first define a test case in Def. 6. We propose to select the executions that reach a final symbolic location with an empty stack, for producing a set of nominal test cases. For this, we select the $\pi$-transitions in *RA* that go from an initial symbolic location to a final one. Any path of each selected transition is then turned into a test case by adding, from each state it reaches, the corresponding transitions that lead to $fail$ in the tester. The last state reached is a final state with empty stack. It is replaced by the verdict *pass*. The non-verdict nodes are 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 the verdict $stack\_fail$. For each state $(l, v, p)$ in each test case, for every action $a^- \in \Gamma^-$, for every coarsest partition $u$ where $\Delta_{a^-}((l, v), u) \neq \varnothing$, if the symbol $a$ is not on the top of $p$, then the transition $((l, v, p), a^-, u, \{y\}, stack\_fail)$ is added to the test case. Thus, the result is a set of test cases, in which the actions are observable (the stack and output actions). Figure 5 shows the two test cases issued from the $\pi$-transition $((l_0, v_0), \pi, (l_6, v_0))$ of Example 2. For example, if the tester observes the pop of the symbol $Fib_2^-$ from the state $(l_6, v_0, [Fib_1])$, then the verdict $stack\_fail$ is emitted.
The tests generated then have to be executed on the IUT. As *TPAIO* are abstractions, the tests are not in general guaranteed to be instantiable as concrete executions of the $IUT$. This is the case in our example of a recursive program whose evaluation conditions have been abstracted in the *TPAIO*. To select the concretizable test cases and to compute their inputs, we propose to use symbolic execution [10], as a mean for analysing a path and finding the corresponding program inputs. A constraint

solver may also be invoked while executing a given test case [13]. The satisfiability of a constraint can be efficiently evaluated by means of SMT solvers such as Z3 [12]. If the constraint is satisfiable, then the test case is concretizable. The solver also finds a solution for the constraint of this concretizable test case. It represents the concrete inputs that lead to the execution of the test case being considered. For example, the test case of Fig. 5(b) represents the trace $ACFib_1^+ABDFib_1^-Fib_2^+ABDFib_2^-E$, which corresponds to the following successive instructions: int $res_1, res_2$; $n > 1$; $res_1 = Fib(n-1)$; int $res_1, res_2$; $n-1 \leqslant 1$; return $n-1$; $res_1 = Fib(n-2)$; int $res_1, res_2$; $n-2 \leqslant 1$; return $n-2$ and return $res_1 + res_2$. It corresponds to the following path constraint: $n > 1 \wedge n-1 \leqslant 1 \wedge n-2 \leqslant 1$. This constraint is satisfiable and a solution is $n = 2$. Thus, this test case corresponds to $Fib(2)$. In our case, we obtain two test cases which are concretizable. The other test case (see Fig. 5(a)), corresponds to $Fib(0)$ or $Fib(1)$.

## 5 Soundness, Incompleteness and Coverage of the Method

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

### *5.1 Soundness*

**Theorem 1.** *A symbolic location $l_i^T$ is reachable with an empty stack in a TPTIO if there exists a $\pi$-transition $(l_0^T, \pi, l_i^T)$ 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 $R_1$ and $R_2$, that create *RA* transitions only from *TPTIO* ones, are sound. Then we prove that the rules $R_3$ and $R_4$ preserve that soundness.

- $R_1$ case: if there is a transition $(l_1, v_1) \xrightarrow{(l_1,v_1)\xrightarrow{A,g_1,\{y\}}(l_2,v_2)} (l_2, v_2) \in \Delta^R$ then $(l_2, v_2)$ is reachable from $(l_1, v_1)$ in the *TPTIO* because by definition of the *TPTIO*, $v_1 \wedge g_1$ is satisfiable.
- $R_2$ case: if there is a transition $(l_1, v_1) \xrightarrow{(l_1,v_1)\xrightarrow{a^+,g_1,\{y\}}(l_2,v_2)\xrightarrow{a^-,g_2,\{y\}}(l_3,v_3)} (l_3, v_3) \in \Delta^R$ then $(l_3, v_3)$ is reachable from $(l_1, v_1)$ in the *TPTIO* because it is always possible to pop $a$ after $a$ has been pushed and by definition of the *TPTIO*, $v_1 \wedge g_1$ and $v_2 \wedge g_2$ are satisfiable.
- $R_3$ case: if there is a transition $(l_1, v_1) \xrightarrow{(l_1,v_1)\xrightarrow{a^+,g_1,\{y\}}(l_2,v_2)\xrightarrow{\pi}(l_3,v_3)\xrightarrow{a^-,g_3,\{y\}}(l_4,v_4)} (l_4, v_4) \in \Delta^R$ then $(l_4, v_4)$ is reachable from $(l_1, v_1)$ in the *TPTIO* because $(l_3, v_3)$ is reachable from $(l_2, v_2)$ according to the induction hypothesis, and it is always

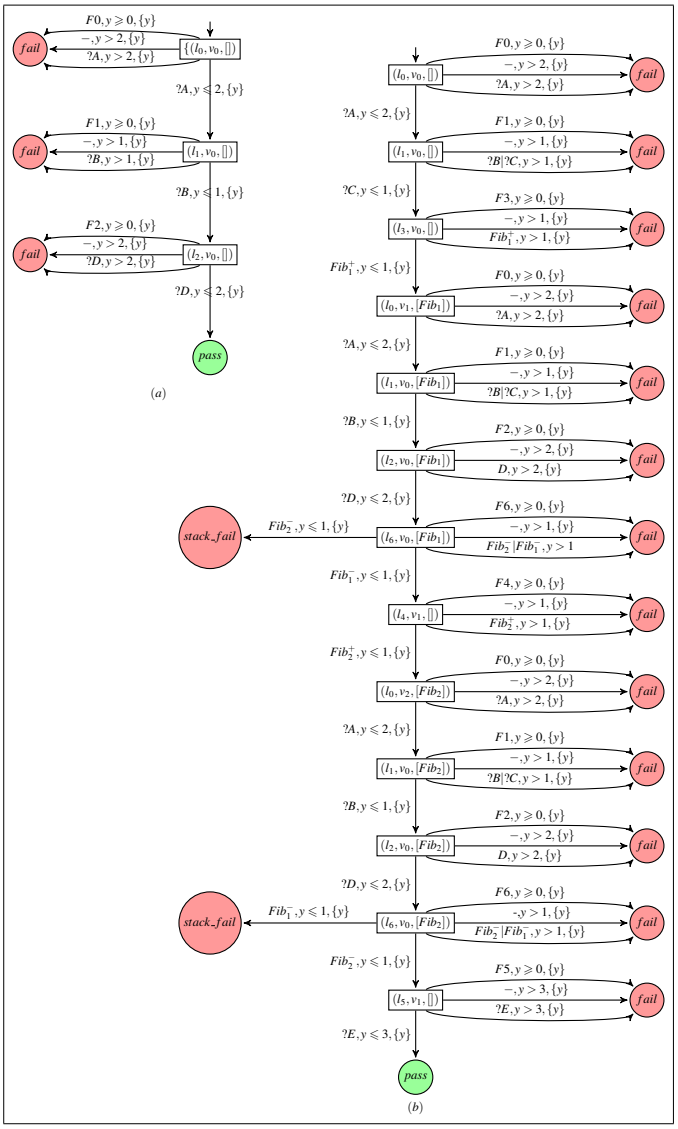


Fig. 5. Two Test Cases of the *TPAIO* of Fig. 2

possible to pop $a$ after $a$ has been pushed, and by definition of the *TPTIO* $v_1 \wedge g_1$ and $v_3 \wedge g_3$ are satisfiable.

- $R_4$ case: if there is a transition $(l_1, v_1) \xrightarrow{\pi_1} (l_2, v_2) \xrightarrow{\pi_2} (l_3, v_3) \in \Delta^R$ then $(l_3, v_3)$ is reachable from $(l_1, v_1)$ in the *TPTIO* because $(l_2, v_2)$ is reachable from

$(l_1, v_1)$ and $(l_3, v_3)$ is reachable from $(l_2, v_2)$, according to the induction hypothesis.

**Theorem 2.** *A location $l_i$ is reachable with an empty stack in a TPAIO iff there exists a $\pi$-transition $((l_0, v_0), \pi, (l_i, v_i))$ in the RA of its TPTIO.*

*Proof.* Let $g$ be the guard of a transition $(l, a, g, X', l')$. A transition $((l, v), a, u, \{y\}, usucc(dsucc((l, v) \cap u, a)))$ is added to $\Delta^T$ where $a \in \Sigma \cup \Gamma^{+-}$ only if $v \wedge u \wedge g$ is satisfiable, by definition of $\Delta_a((l, v), u)$ in Sect. 4.2. Thus the construction of a *TPTIO* from a *TPAIO* takes the clock constraints into account. It preserves both the clock and stack constraints of the *TPAIO*. Thus, Theorem 2 is a direct consequence of Theorem 1.

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

**Theorem 3.** *Let $\pi = (l_0, v_0, p_0) \xrightarrow{a_0, g_0, \{y\}} (l_1, v_1, p_1) \xrightarrow{a_1, g_1, \{y\}} \ldots (l_{n-1}, v_{n-1}, p_{n-1}) \xrightarrow{a_{n-1}, g_{n-1}, \{y\}} (l_n, v_n, p_n) \xrightarrow{a_n, g_n, \{y\}} l_{n+1}$ be a path of a test case of a specification $T = \langle L, l_0, \Sigma_{out} \cup \{\tau\}, \Gamma, X, \Delta, F \rangle$ where $l_i \in L$, $v_i$ is a clock constraint in $CC(X \cup \{y\})$, $g_i \in Grd(\{y\})$, $p_i \in \Gamma^*$, $a_i \in \Sigma_{out} \cup \Gamma^{+-} \cup \{-\}$ 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 $\rho = \delta_0 a_0 \delta_1 a_1 \delta_2 \ldots \delta_{n-1} a_{n-1} \delta_n a_n \in RT(\Sigma_{out} \cup \Gamma^{+-})$ be the trace of the path $\pi$. $(l_n, v_n, p_n)$ is the current symbolic location after the execution of $\delta_0 a_0 \delta_1 a_1 \delta_2 \ldots \delta_{n-1} a_{n-1} \delta_n$ and $g_n$ is the coarsest partition computed such that $\delta_n \in g_n$. 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, according to item $(i)$ in Sect. 4.2. If $\Delta_{a_n}((l_n, v_n), u_n) = \varnothing$, then the transition $((l_n, v_n), a_n, \{y\}, fail)$ is a transition of the *TPTIO*. Therefore, $a_n \notin out(T\ after\ \delta_0 a_0 \delta_1 a_1 \delta_2 \ldots \delta_{n-1} a_{n-1} \delta_n)$, and $I$ does not conform to $T$.
- $fail$ occurs after a timeout $\delta_n$ ($a_n = -$) has been observed, according to item $(ii)$ in Sect. 4.2. Therefore, $v_n + \delta_n \notin out(T\ after\ \delta_0 a_0 \delta_1 a_1 \delta_2 \ldots \delta_{n-1} a_{n-1})$, and $I$ does not conform to $T$.
- $stack\_fail$ occurs after a pop action $a_n$ has been observed, acceptable by the specification ($\Delta_{a_n}((l_n, v_n), u_n) \neq \varnothing$), but in a context where the symbol $a$ should not be on top of the stack $p_n$. 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*).

## 5.2 Incompleteness

The equivalence relation used to compute a *TPTIO* from a *TPAIO* can lead to a loss of precision. It should be possible to build more precise test cases than the

ones computed by our method. Consider for example the *TPAIO* of Fig. 6(a). The *TPTIO* of Fig. 6(c) is more precise than the one of Fig. 6(b). The trace $0a^+2a^-$ leads to the symbolic location $(l_2, 0 \leqslant x-y < 4)$ in the *TPTIO* of Fig. 6(b). It leads to the symbolic location $(l_2, 0 \leqslant x-y \leqslant 3)$ in the *TPTIO* of Fig. 6(c), but not to the symbolic location $(l_2, 0 < x-y < 4)$. We remark that the symbolic location $(l_2, 0 \leqslant x-y \leqslant 3)$ is more precise than $(l_2, 0 \leqslant x-y < 4)$.

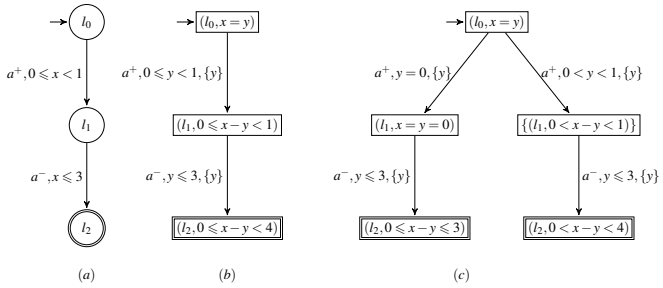


Fig. 6. Two *TPTIO* (b) and (c) of a *TPAIO* (a) (without $fail$ location). (c) is more precise than (b)

### *5.3 Coverage*

In Sect. 4.3, we have presented a method for computing an *RA* from a *TPTIO*. The algorithm that computes the *RA* takes into account the coverage of the transitions of the *TPTIO*. It adds a new $\pi$-transition $(l^T, \pi, l^{T'})$ only if $\pi$ covers a new transition w.r.t the other $\pi$'-transitions $(l^T, \pi', l^{T'})$. The paths of all the $\pi$-transitions that go to a final symbolic location of the *RA* cover all the transitions of the *TPTIO*. But since some test cases might be unconcretizable, the set of concrete test cases is not guaranteed to cover all the transitions of the *TPAIO*. When all the tests are concretizable (it is the case in our example), then all the reachable locations and all the reachable transitions of the *TPAIO* are covered.

## 6 Conclusion and Further Works

We have presented a method to generate test from *TPAIO*. To our knowledge, this had not been treated before in the literature. Our method proceeds by computing reachable locations and transitions, and generating off-line tests from a deterministic *TPAIO* that models a timed recursive program. The tester observes the stack and output actions, as well as the delays. Our method first adapts the tester computation method of [11] for *TA* to the *TPAIO* case. We obtain a *TPTIO* that is a *TPAIO* with only one clock. Its locations are defined as being either rejecting ($fail$) or

symbolic locations. In a second step, we adapt another algorithm presented in [9] for *PA*, for computing the *RA* of the *TPTIO*. We compute the paths of the *TPTIO* associated to 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 *TPTIO*, test cases are generated. If they are concretizable, they cover all the reachable locations and transitions of the *TPAIO*.

Our work is currently for deterministic timed pushdown automata with outputs. We intend as a future work to generalize to non-deterministic timed pushdown automata with inputs and outputs, to target general timed recursive systems. Also, at this stage of our work, we have developed a proof-of-concept prototype to experimentally validate our approach, in which the test generation process have been automated. We intend as a future work to develop this tool, in order to be able to perform larger scale experiments.

## References


1. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-timed pushdown automata. In: LICS, pp. 35–44 (2012)
2. Alur, R., Dill, D.L.: A theory of timed automata. TCS **126**(2), 183–235 (1994)
3. Autebert, J.M., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: Handbook of Formal Languages, vol. 1, pp. 111–174. Springer (1997)
4. Benerecetti, M., Minopoli, S., Peron, A.: Analysis of timed recursive state machines. In: TIME 2010, pp. 61–68. IEEE (2010)
5. Bouajjani, A., Echahed, R., Robbana, R.: On the automatic verification of systems with continuous variables and unbounded discrete data structures. In: Hybrid Systems II, *LNCS*, vol. 999, pp. 64–85 (1995)
6. Cassez, F., Jessen, J.J., Larsen, K.G., Raskin, J.F., Reynier, P.A.: Automatic synthesis of robust and optimal controllers — an industrial case study. In: HSCC '09, pp. 90–104. Springer (2009)
7. Chadha, R., Legay, A., Prabhakar, P., Viswanathan, M.: Complexity bounds for the verification of real-time software. In: VMCAI'10, *LNCS*, vol. 5944, pp. 95–111. Springer (2010)
8. Dreyfus, A., Héam, P.C., Kouchnarenko, O., Masson, C.: A random testing approach using pushdown automata. STVR **24**(8), 656–683 (2014)
9. Finkel, A., Willems, B., Wolper, P.: A direct symbolic approach to model checking pushdown systems (ext. abs.). In: Infinity, *ENTCS*, vol. 9, pp. 27–37 (1997)
10. Godefroid, P.: Test generation using symbolic execution. In: IARCS, pp. 24–33 (2012)
11. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. FMSD **34**(3), 238–304 (2009)
12. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS, *LNCS*, vol. 4963, pp. 337–340 (2008)
13. Păsăreanu, C.S., Rungta, N., Visser, W.: Symbolic execution with mixed concrete-symbolic solving. In: ISSTA'11, pp. 34–44. ACM (2011)
14. Sénizergues, G.: L(a) = l(b)? decidability results from complete formal systems. In: ICALP, *LNCS*, vol. 2380, pp. 1–37. Springer (2002)
15. Tretmans, J.: Testing concurrent systems: A formal approach. In: Concurrency Theory, *LNCS*, vol. 1664, pp. 46–65. Springer (1999)
16. Trivedi, A., Wojtczak, D.: Recursive timed automata. In: ATVA'10, *LNCS*, vol. 6252, pp. 306–324. Springer (2010)