



HAL
open science

Off-line Test Selection with Test Purposes for Non-Deterministic Timed Automata

Nathalie Bertrand, Thierry Jéron, Amélie Stainer, Moez Krichen

► **To cite this version:**

Nathalie Bertrand, Thierry Jéron, Amélie Stainer, Moez Krichen. Off-line Test Selection with Test Purposes for Non-Deterministic Timed Automata. Logical Methods in Computer Science, 2012, 8 (4:8), pp.1-33. hal-00744074

HAL Id: hal-00744074

<https://inria.hal.science/hal-00744074>

Submitted on 22 Oct 2012

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.

OFF-LINE TEST SELECTION WITH TEST PURPOSES FOR NON-DETERMINISTIC TIMED AUTOMATA *

NATHALIE BERTRAND ^a, THIERRY JÉRON ^b, AMÉLIE STAINER ^c, AND MOEZ KRICHEN ^d

^{a,b} Inria Rennes - Bretagne Atlantique, Rennes, France
e-mail address: {nathalie.bertrand, thierry.jeron}@inria.fr

^c University of Rennes 1, Rennes, France
e-mail address: amelie.stainer@inria.fr

^d University of Sfax, Tunisia
e-mail address: moez.krichen@redcad.org

ABSTRACT. This article proposes novel off-line test generation techniques from non-deterministic timed automata with inputs and outputs (TAIOs) in the formal framework of the **tioco** conformance theory. In this context, a first problem is the determinization of TAIOs, which is necessary to foresee next enabled actions after an observable trace, but is in general impossible because not all timed automata are determinizable. This problem is solved thanks to an approximate determinization using a game approach. The algorithm performs an io-abstraction which preserves the **tioco** conformance relation and thus guarantees the soundness of generated test cases. A second problem is the selection of test cases from a TAIO specification. The selection here relies on a precise description of timed behaviors to be tested which is carried out by expressive test purposes modeled by a generalization of TAIOs. Finally, an algorithm is described which generates test cases in the form of TAIOs equipped with verdicts, using a symbolic co-reachability analysis guided by the test purpose. Properties of test cases are then analyzed with respect to the precision of the approximate determinization: when determinization is exact, which is the case on known determinizable classes, in addition to soundness, properties characterizing the adequacy of test cases verdicts are also guaranteed.

INTRODUCTION

Conformance testing is the process of testing whether some implementation of a software system behaves correctly with respect to its specification. In this testing framework, implementations are considered as *black boxes*, *i.e.* the source code is unknown, only their interface with the environment is known and used to interact with the tester. In *formal model-based*

1998 ACM Subject Classification: D.2.4, D.2.5, D.4.7, F.1.1.

Key words and phrases: Conformance testing, timed automata, partial observability, urgency, approximate determinization, game, test purpose.

* This article is based on the material of the TACAS conference paper [5].

This work was partly funded by the French project TESTEC (ANR-07-TLOG-022).

conformance testing, models are used to describe testing artifacts (specifications, implementations, test cases, ...). Moreover, conformance is formally defined as a relation between implementations and specifications which reflects what are the correct behaviors of the implementation with respect to those of the specification. Defining such a relation requires the hypothesis that the implementation behaves as a model. Test cases with verdicts, which will be executed against the implementation in order to check conformance, are generated automatically from the specification. Test generation algorithms should then ensure properties relating verdicts of executions of test cases with the conformance relation (*e.g.* soundness), thus improving the quality of testing compared to manual writing of test cases.

For timed systems, model-based conformance testing has already been explored in the last decade, with different models and conformance relations (see *e.g.* [22] for a survey), and various test generation algorithms (*e.g.* [8, 18, 21]). In this context, a very popular model is *timed automata with inputs and outputs* (TAIOs), a variant of *timed automata* (TAs) [1], in which the alphabet of observable actions is partitioned into inputs and outputs. We consider here a very general model, partially observable and non-deterministic TAIOs with invariants for the modeling of urgency. We resort to the **tioco** conformance relation defined for TAIOs [17], which is equivalent to the **rtioco** relation [19]. This relation compares the observable behaviors of timed systems, made of inputs, outputs and delays, restricting attention to what happens after specification traces. Intuitively, an implementation conforms to a specification if after any observable trace of the specification, outputs and delays observed on the implementation after this trace should be allowed by the specification.

One of the main difficulties encountered in test generation for those partially observable, non-deterministic TAIOs is determinization. In fact determinization is required in order to foresee the next enabled actions during execution, and thus to emit a correct verdict depending on whether actions observed on the implementation are allowed by the specification model after the current observable behavior. Unfortunately, TAs (and thus TAIOs) are not determinizable in general [1]: the class of deterministic TAs is a strict subclass of TAs. Two different approaches have been taken for test generation from timed models, which induce different treatments of non-determinism.

- In *off-line test generation* test cases are first generated as timed automata (or timed sequences, or timed transition systems) and subsequently executed on the implementation. One advantage is that test cases can be stored and further used *e.g.* for regression testing and serve for documentation. However, due to the non-determinizability of TAIOs, the approach has often been limited to deterministic or determinizable TAIOs (see *e.g.* [15, 21]). A notable exception is [18] where the problem is solved by the use of an over-approximate determinization with fixed resources (number of clocks and maximal constant): a deterministic automaton with those resources is built, which simulates the behaviors of the non-deterministic one. Another one is [10] where winning strategies of timed games are used as test cases.
- In *on-line test generation*, test cases are generated during their execution. After the current observed trace, enabled actions after this trace are computed from the specification model and, either an allowed input is sent to the implementation, or a received output or an observed delay is checked. This technique can be applied to any TAIO, as possible observable actions are computed only along the current finite execution (the set of possible states of the specification model after a finite trace, and their enabled actions are finitely representable and computable), thus avoiding a complete determinization. On-line test generation is of particular interest to rapidly discover errors, can be applied to large and

non-deterministic systems, but may sometimes be impracticable due to a lack of reactivity (the time needed to compute successor states on-line may sometimes be incompatible with real-time constraints).

Our feeling is that off-line test generation from timed models did not receive much attention because of the inherent difficulty of determinization. However, recent works on approximate determinization of timed automata [18, 7] open the way to new research approaches and results in this domain.

Contribution. In this paper, we propose to generate test cases off-line for the whole class of non-deterministic TAIOS, in the formal context of the **tioco** conformance theory. The determinization problem is tackled thanks to an approximate determinization with fixed resources in the spirit of [18], using a game approach allowing to more closely simulate the non-deterministic TAIOS [7]. Our approximate determinization method is more precise than [18] (see [7, 6] for details), preserves the richness of our model by dealing with partial observability and urgency, and can be adapted to testing by a different treatment of inputs, outputs and delays. Determinization is exact for known classes of determinizable TAIOS (*e.g.* event-clock TAs, TAs with integer resets, strongly non-Zeno TAs) if resources are sufficient. In the general case, determinization may over-approximate outputs and delays and under-approximate inputs. More precisely, it produces a deterministic *io-abstraction* of the TAIOS for a particular *io-refinement* relation which generalizes the one of [9]. As a consequence, if test cases are generated from the io-abstract deterministic TAIOS and are sound for this TAIOS, they are guaranteed to be sound for the original (io-refined) non-deterministic TAIOS.

Behaviors of specifications to be tested are identified by means of test purposes. Test purposes are often used in testing practice, and are particularly useful when one wants to focus testing on particular behaviors, *e.g.* corresponding to requirements or suspected behaviors of the implementation. In this paper they are defined as *open timed automata with inputs and outputs* (OTAIOS), a model generalizing TAIOS, allowing to precisely target some behaviors according to actions and clocks of the specification as well as proper clocks. Then, in the same spirit as for the TGV tool in the untimed case [13], test selection is performed by a construction relying on a co-reachability analysis. Produced test cases are in the form of TAIOS, while most approaches generate less elaborated test cases in the form of timed traces or trees. In addition to soundness, when determinization is exact, we also prove an exhaustiveness property, and two other properties on the adequacy of test case verdicts. To our knowledge, this whole work constitutes the most general and advanced off-line test selection approach for TAIOS.

This article is a long version of [5]. In addition to the proofs of key properties, it also contains much more details, explanations, illustrations by examples, complexity considerations, and a new result on exhaustiveness of the test generation method.

Outline. The paper is structured as follows. In the next section we introduce the model of OTAIOS, its semantics, some notations and operations on this model and the model of TAIOS. Section 2 recalls the **tioco** conformance theory for TAIOS, including properties of test cases relating conformance and verdicts, and introduces an io-refinement relation which preserves **tioco**. Section 3 presents our game approach for the approximate determinization compatible with the io-refinement. In Section 4 we detail the test selection mechanism using

test purposes and prove some properties on generated test cases. Section 5 discusses some issues related to test case execution and test purposes and some related work.

1. A MODEL OF OPEN TIMED AUTOMATA WITH INPUTS/OUTPUTS

Timed automata (TAs) [1] is a usual model for time constrained systems. In the context of model-based testing, TAs have been extended to timed automata with inputs and outputs (TAIOs) whose sets of actions are partitioned into inputs, outputs and unobservable actions. In this section, we further extend TAIOs by partitioning the set of clocks into proper clocks (*i.e.*, controlled by the automaton) and observed clocks (*i.e.*, owned by some other automaton). The resulting model of *open timed automata with inputs/outputs* (OTAIOs for short), allows one to describe observer timed automata that can test clock values from other automata. While the sub-model of TAIOs (with only proper clocks) is sufficient for most testing artifacts (specifications, implementations, test cases) observed clocks of OTAIOs will be useful to express test purposes whose aim is to focus on the timed behaviors of the specification. Like in the seminal paper for TAs [1], we consider OTAIOs and TAIOs with location invariants to model urgency.

1.1. Timed automata with inputs/outputs. We start by introducing notations and useful definitions concerning TAIOs and OTAIOs.

Given X a finite set of *clocks*, a *clock valuation* is a mapping $v : X \rightarrow \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers. $\bar{0}$ stands for the valuation assigning 0 to all clocks. If v is a valuation over X and $t \in \mathbb{R}_{\geq 0}$, then $v + t$ denotes the valuation which assigns to every clock $x \in X$ the value $v(x) + t$. For $X' \subseteq X$ we write $v_{[X' \leftarrow 0]}$ for the valuation equal to v on $X \setminus X'$ and assigning 0 to all clocks of X' . Given M a non-negative integer, an *M-bounded guard* (or simply *guard*) over X is a finite conjunction of constraints of the form $x \sim c$ where $x \in X$, $c \in [0, M] \cap \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. Given g a guard and v a valuation, we write $v \models g$ if v satisfies g . We sometimes abuse notations and write g for the set of valuations satisfying g . Invariants are restricted cases of guards: given $M \in \mathbb{N}$, an *M-bounded invariant* over X is a finite conjunction of constraints of the form $x \triangleleft c$ where $x \in X, c \in [0, M] \cap \mathbb{N}$ and $\triangleleft \in \{<, \leq\}$. We denote by $G_M(X)$ (resp. $I_M(X)$) the set of M -bounded guards (resp. invariants) over X .

In the sequel, we write \sqcup for the disjoint union of sets, and use it, when appropriate, to insist on the fact that sets are disjoint.

Definition 1.1 (OTAIO). An *open timed automaton with inputs and outputs* (OTAIO) is a tuple $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_{\tau}^{\mathcal{A}}, \Sigma_1^{\mathcal{A}}, \Sigma_2^{\mathcal{A}}, X_p^{\mathcal{A}}, X_o^{\mathcal{A}}, M^{\mathcal{A}}, I^{\mathcal{A}}, E^{\mathcal{A}})$ such that:

- $L^{\mathcal{A}}$ is a finite set of *locations*, with $\ell_0^{\mathcal{A}} \in L^{\mathcal{A}}$ the *initial location*,
- $\Sigma_{\tau}^{\mathcal{A}}$, $\Sigma_1^{\mathcal{A}}$ and $\Sigma_2^{\mathcal{A}}$ are disjoint finite *alphabets* of *input actions* (noted $a?, b?, \dots$), *output actions* (noted $a!, b!, \dots$), and *internal actions* (noted τ_1, τ_2, \dots). We note $\Sigma_{obs}^{\mathcal{A}} = \Sigma_{\tau}^{\mathcal{A}} \sqcup \Sigma_1^{\mathcal{A}}$ for the alphabet of observable actions, and $\Sigma^{\mathcal{A}} = \Sigma_{\tau}^{\mathcal{A}} \sqcup \Sigma_1^{\mathcal{A}} \sqcup \Sigma_2^{\mathcal{A}}$ for the whole set of actions.
- $X_p^{\mathcal{A}}$ and $X_o^{\mathcal{A}}$ are disjoint finite sets of *proper clocks* and *observed clocks*, respectively. We note $X^{\mathcal{A}} = X_p^{\mathcal{A}} \sqcup X_o^{\mathcal{A}}$ for the whole set of *clocks*.
- $M^{\mathcal{A}} \in \mathbb{N}$ is the *maximal constant* of \mathcal{A} , and we will refer to $(|X^{\mathcal{A}}|, M^{\mathcal{A}})$ as the *resources* of \mathcal{A} ,

- $I^A : L^A \rightarrow I_{M^A}(X^A)$ is a mapping which labels each location with an M -bounded invariant,
- $E^A \subseteq L^A \times G_{M^A}(X^A) \times \Sigma^A \times 2^{X_p^A} \times L^A$ is a finite set of *edges* where *guards* are defined on X^A , but *resets* are restricted to proper clocks in X_p^A .

One of the reasons for introducing the OTAIO model is to have a uniform model (syntax and semantics) that will be next specialized for particular testing artifacts. In particular, an OTAIO with an empty set of observed clocks X_o^A is a classical TAIIO, and will be the model for specifications, implementations and test cases. The partition of actions reflects their roles in the testing context: the tester cannot observe internal actions, but controls inputs and observes outputs (and delays). The set of clocks is also partitioned into *proper clocks*, *i.e.* usual clocks controlled by the system itself through resets, as opposed to *observed clocks* referring to proper clocks of another OTAIO (*e.g.* modeling the system's environment). These cannot be reset to avoid intrusiveness, but synchronization with them in guards and invariants is allowed. This partition of clocks will be useful for test purposes which can have, as observed clocks, some proper clocks of specifications, with the aim of selecting time constrained behaviors of specifications to be tested.

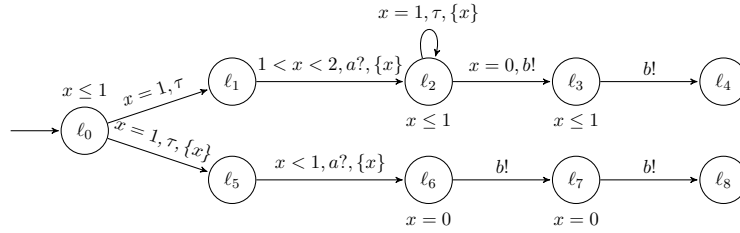


Figure 1: Specification \mathcal{A}

Example 1.2. Figure 1 represents a TAIIO for a specification \mathcal{A} that will serve as a running example in this paper. Its clocks are $X = X_p^A = \{x\}$, its maximal constant is $M^A = 2$, it has a single input $\Sigma_7^A = \{a\}$, a single output $\Sigma_1^A = \{b\}$ and one internal action $\Sigma_\tau^A = \{\tau\}$. Informally, its behavior is as follows. It may stay in the initial location ℓ_0 while $x \leq 1$, and at $x = 1$, has the choice, either to go to ℓ_1 with action τ , or go to ℓ_5 with action τ while resetting x . In ℓ_1 , it may receive a and move to ℓ_2 when x is between 1 and 2, and reset x . In ℓ_2 it may stay while $x \leq 1$ and, either send b and go to ℓ_3 at $x = 0$, or loop silently when $x = 1$ while resetting x . This means that b can be sent at any integer delay after entering ℓ_2 . In ℓ_3 it may stay while $x \leq 1$ and move to ℓ_4 when sending b . In ℓ_5 , one can move to ℓ_6 before $x = 1$ by receiving a and resetting x . Due to invariants $x = 0$ in ℓ_6 and ℓ_7 , the subsequent behavior consists in the immediate transmission of two b 's.

1.2. The semantics of OTAIOs. Let $\mathcal{A} = (L^A, \ell_0^A, \Sigma_7^A, \Sigma_1^A, \Sigma_\tau^A, X_p^A, X_o^A, M^A, I^A, E^A)$ be an OTAIO. The semantics of \mathcal{A} is a *timed transition system* $\mathcal{T}^A = (S^A, s_0^A, \Gamma^A, \rightarrow_{\mathcal{A}})$ where

- $S^A = L^A \times \mathbb{R}_{\geq 0}^{X^A}$ is the set of *states* *i.e.* pairs (ℓ, v) consisting in a location and a valuation of clocks;
- $s_0^A = (\ell_0^A, \bar{0}) \in S^A$ is the *initial state*;

- $\Gamma^A = \mathbb{R}_{\geq 0} \sqcup E^A \times 2^{X_o^A}$ is the set of transition *labels* consisting in either a delay δ or a pair (e, X'_o) formed by an edge $e \in E$ and a set $X'_o \subseteq X_o^A$ of observed clocks;
- the transition relation $\rightarrow_{\mathcal{A}} \subseteq S^A \times \Gamma^A \times S^A$ is the smallest set of the following moves:
 - *Discrete moves*: $(\ell, v) \xrightarrow{(e, X'_o)}_{\mathcal{A}} (\ell', v')$ whenever there exists $e = (\ell, g, a, X'_p, \ell') \in E^A$ such that $v \models g \wedge I^A(\ell)$, $X'_o \subseteq X_o^A$ is an arbitrary subset of observed clocks, $v' = v_{[X'_p \sqcup X'_o \leftarrow 0]}$ and $v' \models I^A(\ell')$. Note that X'_o is unconstrained as observed clocks are not controlled by \mathcal{A} but by a peer OTAIO.
 - *Time elapse*: $(\ell, v) \xrightarrow{\delta}_{\mathcal{A}} (\ell, v + \delta)$ for $\delta \in \mathbb{R}_{\geq 0}$ if $v + \delta \models I^A(\ell)$.

The semantics of OTAIOs generalizes the usual semantics of TAIOS. The difference lies in the treatment of the additional observed clocks as the evolution of those clocks is controlled by a peer OTAIO. The observed clocks evolve at the same speed as the proper clocks, thus continuous moves are simply extended to proper and observed clocks. For discrete moves however, resets of observed clocks are uncontrolled, thus all possible resets have to be considered.

A *partial run* of \mathcal{A} is a finite sequence of subsequent moves in $(S^A \times \Gamma^A)^*.S^A$. For example $\rho = s_0 \xrightarrow{\delta_1}_{\mathcal{A}} s'_0 \xrightarrow{(e_1, X_o^1)}_{\mathcal{A}} s_1 \cdots s_{k-1} \xrightarrow{\delta_k}_{\mathcal{A}} s'_{k-1} \xrightarrow{(e_k, X_o^k)}_{\mathcal{A}} s_k$. The sum of delays in ρ is noted $time(\rho)$. A *run* is a partial run starting in s_0^A . A state s is *reachable* if there exists a run leading to s . A state s is *co-reachable* from a set $S' \subseteq S^A$ if there is a partial run from s to a state in S' . We note $\mathbf{reach}(\mathcal{A})$ the set of reachable states and $\mathbf{coreach}(\mathcal{A}, S')$ the set of states co-reachable from S' .

A (partial) *sequence* is a projection of a (partial) run where states are forgotten, and discrete transitions are abstracted to actions and proper resets which are grouped with observed resets. As an example, the sequence corresponding to a run

$$\rho = s_0 \xrightarrow{\delta_1}_{\mathcal{A}} s'_0 \xrightarrow{(e_1, X_o^1)}_{\mathcal{A}} s_1 \cdots s_{k-1} \xrightarrow{\delta_k}_{\mathcal{A}} s'_{k-1} \xrightarrow{(e_k, X_o^k)}_{\mathcal{A}} s_k$$

is

$$\mu = \delta_1.(a_1, X_p^1 \sqcup X_o^1) \cdots \delta_k.(a_k, X_p^k \sqcup X_o^k)$$

where $e_i = (\ell_i, g_i, a_i, X_p^i, \ell'_i)$ for all $i \in [1, k]$. We then note $s_0 \xrightarrow{\mu}_{\mathcal{A}} s_k$. We write $s_0 \xrightarrow{\mu}_{\mathcal{A}}$ if there exists s_k such that $s_0 \xrightarrow{\mu}_{\mathcal{A}} s_k$. We note $\mathbf{Seq}(\mathcal{A}) \subseteq (\mathbb{R}_{\geq 0} \sqcup (\Sigma^A \times 2^{X^A}))^*$ (respectively $\mathbf{pSeq}(\mathcal{A})$) the set of sequences (resp. partial sequences) of \mathcal{A} . For a sequence μ , $time(\mu)$ denotes the sum of delays in μ .

For a (partial) sequence $\mu \in \mathbf{pSeq}(\mathcal{A})$, $Trace(\mu) \in (\mathbb{R}_{\geq 0} \sqcup \Sigma_{obs}^A)^*.\mathbb{R}_{\geq 0}$ denotes the observable behavior obtained by erasing internal actions and summing delays between observable ones. It is defined inductively as follows:

- $Trace(\varepsilon) = 0$,
- $Trace(\delta_1 \dots \delta_k) = \sum_{i=1}^k \delta_i$,
- $Trace(\delta_1 \dots \delta_k.(\tau, X').\mu) = Trace((\sum_{i=1}^k \delta_i).\mu)$,
- $Trace(\delta_1 \dots \delta_k.(a, X').\mu) = (\sum_{i=1}^k \delta_i).a.Trace(\mu)$ if $a \in \Sigma_{obs}^A$.

For example $Trace(1.(\tau, X^1).2.(a, X^2).2.(\tau, X^3)) = 3.a.2$ and $Trace(1.(\tau, X^1).2.(a, X^2)) = 3.a.0$. When a trace ends by a 0-delay, we sometimes omit it and write e.g. $3.a$ for $3.a.0$.

When concatenating two traces, the last delay of the first trace and the initial delay of the second one must be added up as follows: if $\sigma_1 = \delta_1.a_1 \cdots a_n.\delta_{n+1}$ and $\sigma_2 =$

$\delta'_1.a'_1 \cdots a'_m.\delta'_{m+1}$ then $\sigma_1.\sigma_2 = \delta_1.a_1 \cdots a_n.(\delta_{n+1} + \delta'_1).a'_1 \cdots a'_m.\delta'_{m+1}$. Concatenation allows one to define the notion of prefix. Given a trace σ , σ_1 is a *prefix* of σ if there exists some σ_2 with $\sigma = \sigma_1.\sigma_2$. Under this definition, 1.a.1 is a prefix of 1.a.2.b.

For a run ρ projecting onto a sequence μ , we also write $Trace(\rho)$ for $Trace(\mu)$. The set of traces of runs of \mathcal{A} is denoted by $Traces(\mathcal{A}) \subseteq (\mathbb{R}_{\geq 0} \sqcup \Sigma_{obs}^{\mathcal{A}})^*.\mathbb{R}_{\geq 0}$ ¹.

Two OTAIOS are said *equivalent* if they have the same sets of traces.

Let $\sigma \in (\mathbb{R}_{\geq 0} \sqcup \Sigma_{obs}^{\mathcal{A}})^*.\mathbb{R}_{\geq 0}$ be a trace, and $s \in S^{\mathcal{A}}$ be a state,

- \mathcal{A} **after** $\sigma = \{s \in S^{\mathcal{A}} \mid \exists \mu \in \text{Seq}(\mathcal{A}), s_0^{\mathcal{A}} \xrightarrow{\mu}_{\mathcal{A}} s \wedge Trace(\mu) = \sigma\}$ denotes the set of states where \mathcal{A} can stay after observing the trace σ .
- $elapse(s) = \{t \in \mathbb{R}_{\geq 0} \mid \exists \mu \in (\mathbb{R}_{\geq 0} \sqcup (\Sigma_{\tau}^{\mathcal{A}} \times 2^{X^{\mathcal{A}}}))^*, s \xrightarrow{\mu}_{\mathcal{A}} \wedge time(\mu) = t\}$ is the set of enabled delays in s with no observable action.
- $out(s) = \{a \in \Sigma_1^{\mathcal{A}} \mid \exists X \subseteq X^{\mathcal{A}}, s \xrightarrow{(a,X)}_{\mathcal{A}}\} \cup \text{elapse}(s)$ (and $in(s) = \{a \in \Sigma_2^{\mathcal{A}} \mid s \xrightarrow{(a,X)}_{\mathcal{A}}\}$) for the set of outputs and delays (respectively inputs) that can be observed from s . For $S' \subseteq S^{\mathcal{A}}$, $out(S') = \bigcup_{s \in S'} out(s)$ and $in(S') = \bigcup_{s \in S'} in(s)$.

Using these last definitions, we will later describe the set of possible outputs and delays after the trace σ by $out(\mathcal{A} \text{ after } \sigma)$.

Notice that all notions introduced for OTAIOS apply to the subclass of TAIOS.

1.3. Properties and operations. A TAIOS \mathcal{A} is *deterministic* (and called a DTAIO) whenever for any $\sigma \in Traces(\mathcal{A})$, \mathcal{A} **after** σ is a singleton². A TAIOS \mathcal{A} is *determinizable* if there exists an equivalent DTAIO. It is well-known that some timed automata are not determinizable [1]; moreover, the determinizability of timed automata is an undecidable problem, even with fixed resources [24, 12].

An OTAIOS \mathcal{A} is said *complete* if in every location ℓ , $I^{\mathcal{A}}(\ell) = \text{true}$ and for every action $a \in \Sigma^{\mathcal{A}}$, the disjunction of all guards of transitions leaving ℓ and labeled by a is **true**. This entails that $\text{Seq}(\mathcal{A}) \downarrow_{X_p^{\mathcal{A}}} = (\mathbb{R}_{\geq 0} \sqcup (\Sigma^{\mathcal{A}} \times 2^{X_o^{\mathcal{A}}}))^*$, where $\downarrow_{X_p^{\mathcal{A}}}$ is the projection that removes resets of proper clocks in $X_p^{\mathcal{A}}$. This means that \mathcal{A} is universal for all the behaviors of its environment.

An OTAIOS \mathcal{A} is *input-complete* in a state $s \in \text{reach}(\mathcal{A})$, if $in(s) = \Sigma_2^{\mathcal{A}}$. An OTAIOS \mathcal{A} is input-complete if it is input-complete in all its reachable states.

An OTAIOS \mathcal{A} is *non-blocking* if $\forall s \in \text{reach}(\mathcal{A}), \forall t \in \mathbb{R}_{\geq 0}, \exists \mu \in \text{pSeq}(\mathcal{A}) \cap (\mathbb{R}_{\geq 0} \sqcup ((\Sigma_1^{\mathcal{A}} \sqcup \Sigma_{\tau}^{\mathcal{A}}) \times 2^{X^{\mathcal{A}}}))^*, time(\mu) = t \wedge s \xrightarrow{\mu}_{\mathcal{A}}$. This means that it never blocks the evolution of time, waiting for an input.

For modeling the behavior of composed systems, in particular for modeling the execution of test cases on implementations, we introduce the classical parallel product. This operation consists in the synchronization of two TAIOS on complementary observable actions (*e.g.* $a!$, the emission of a and $a?$ its reception) and induces the intersection of the sets of traces. It is only defined for *compatible* TAIOS, *i.e.* $\mathcal{A}^i = (L^i, \ell_0^i, \Sigma_2^i, \Sigma_1^i, \Sigma_{\tau}^i, X_p^i, M^i, I^i, E^i)$ for $i = 1, 2$ such that $\Sigma_1^1 = \Sigma_2^2$, $\Sigma_2^1 = \Sigma_1^2$, $\Sigma_{\tau}^1 \cap \Sigma_{\tau}^2 = \emptyset$ and $X_p^1 \cap X_p^2 = \emptyset$.

¹Notice that formally, a trace always ends with a delay, which can be 0. This technical detail is useful later to define verdicts as soon as possible without waiting for a hypothetical next action.

²Determinism is only defined (and used in the sequel) for TAIOS. For OTAIOS, the right definition would consider the projection of \mathcal{A} **after** σ which forgets values of observed clocks, as these introduce “environmental” non-determinism.

Definition 1.3 (Parallel product). The *parallel product* of two compatible TAIOS $\mathcal{A}^i = (L^i, \ell_0^i, \Sigma_\gamma^i, \Sigma_1^i, \Sigma_\tau^i, X_p^i, M^i, I^i, E^i)$ $i = 1, 2$ is a TAIIO $\mathcal{A}^1 \parallel \mathcal{A}^2 = (L, \ell_0, \Sigma_\gamma, \Sigma_1, \Sigma_\tau, X_p, M, I, E)$ where:

- $L = L^1 \times L^2$, $\ell_0 = (\ell_0^1, \ell_0^2)$,
- $\Sigma_\gamma = \Sigma_\gamma^1 \sqcup \Sigma_\gamma^2$, $\Sigma_1 = \Sigma_1^1$ and $\Sigma_\tau = \Sigma_\tau^1 \sqcup \Sigma_\tau^2$
- $X_p = X_p^1 \sqcup X_p^2$
- $M = \max(M^1, M^2)$
- $\forall (\ell^1, \ell^2) \in L, I((\ell^1, \ell^2)) = I(\ell^1) \wedge I(\ell^2)$
- E is the smallest relation such that:
 - for $a \in \Sigma_\gamma^1 \sqcup \Sigma_\gamma^2$, if $(\ell^1, g^1, a, X_p^1, \ell^1) \in E^1$ and $(\ell^2, g^2, a, X_p^2, \ell^2) \in E^2$ then $((\ell^1, \ell^2), g^1 \wedge g^2, a, X_p^1 \cup X_p^2, (\ell^1, \ell^2)) \in E$, *i.e.* complementary actions synchronize, corresponding to a communication;
 - for $\tau_1 \in \Sigma_\tau^1$, $\ell^2 \in L^2$, if $(\ell^1, g^1, \tau_1, X_p^1, \ell^1) \in E^1$ then $((\ell^1, \ell^2), g^1, \tau_1, X_p^1, (\ell^1, \ell^2)) \in E$, *i.e.* internal actions of \mathcal{A}_1 progress independently;
 - for $\tau_2 \in \Sigma_\tau^2$, $\ell^1 \in L^1$, if $(\ell^2, g^2, \tau_2, X_p^2, \ell^2) \in E^2$ then $((\ell^1, \ell^2), g^2, \tau_2, X_p^2, (\ell^1, \ell^2)) \in E$, *i.e.* internal actions of \mathcal{A}_2 progress independently.

By the definition of the transition relation E of $\mathcal{A}^1 \parallel \mathcal{A}^2$, TAIOS synchronize exactly on complementary observable actions and time, and evolve independently on internal actions. As a consequence, the following equality on traces holds:

$$\text{Traces}(\mathcal{A}^1 \parallel \mathcal{A}^2) = \text{Traces}(\mathcal{A}^1) \cap \text{Traces}(\mathcal{A}^2) \quad (1.1)$$

Notice that the definition is not absolutely symmetrical, as the direction (input/output) of actions of the product is chosen with respect to \mathcal{A}^1 . The technical reason is that, in the execution of a test case on an implementation, we will need to keep the directions of actions of the implementation.

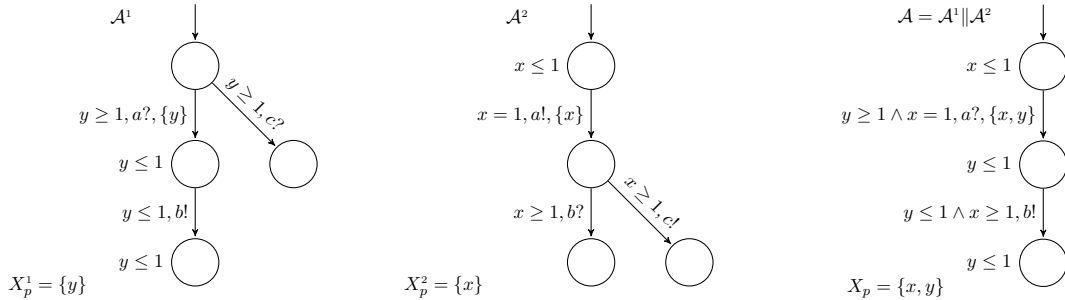


Figure 2: Example of a parallel product $\mathcal{A} = \mathcal{A}^1 \parallel \mathcal{A}^2$.

Example 1.4. The Figure 2 gives a very simple illustration of the parallel product. The intersection of the sets of traces is clear. Indeed, the parallel product recognizes exactly all prefixes of the trace $1.a.1.b$.

We now define a product operation on OTAIOS which extends the classical product of TAs, with a particular attention to observed clocks. This product is used later in the paper, to model the action of a test purpose which observes the clocks of a specification.

Definition 1.5 (Product). Let $\mathcal{A}^i = (L^i, \ell_0^i, \Sigma_\tau, \Sigma_1, \Sigma_\tau, X_p^i, X_o^i, M^i, I^i, E^i)$, $i = 1, 2$, be two OTAIOS with same alphabets and disjoint sets of proper clocks ($X_p^1 \cap X_p^2 = \emptyset$). Their *product* is the OTAIO $\mathcal{A}^1 \times \mathcal{A}^2 = (L, \ell_0, \Sigma_\tau, \Sigma_1, \Sigma_\tau, X_p, X_o, M, I, E)$ where:

- $L = L^1 \times L^2$;
- $\ell_0 = (\ell_0^1, \ell_0^2)$;
- $X_p = X_p^1 \sqcup X_p^2$, $X_o = (X_o^1 \cup X_o^2) \setminus X_p$;
- $M = \max(M^1, M^2)$;
- $\forall (\ell^1, \ell^2) \in L, I((\ell^1, \ell^2)) = I^1(\ell^1) \wedge I^2(\ell^2)$;
- $((\ell^1, \ell^2), g^1 \wedge g^2, a, X_p^1 \sqcup X_p^2, (\ell^1, \ell^2)) \in E$ if $(\ell^i, g^i, a, X_p^i, \ell^i) \in E^i$, $i=1,2$.

Intuitively, \mathcal{A}^1 and \mathcal{A}^2 synchronize on both time and common actions (including internal ones³). \mathcal{A}^2 may observe proper clocks of \mathcal{A}^1 using its observed clocks $X_p^1 \cap X_o^2$, and *vice versa*. The set of proper clocks of $\mathcal{A}^1 \times \mathcal{A}^2$ is the union of proper clocks of \mathcal{A}^1 and \mathcal{A}^2 , and observed clocks of $\mathcal{A}^1 \times \mathcal{A}^2$ are observed clocks of any OTAIO which are not proper. For example, the OTAIO in Figure 13 represents the product of the TAIO \mathcal{A} in Figure 1 and the OTAIO \mathcal{TP} of Figure 12.

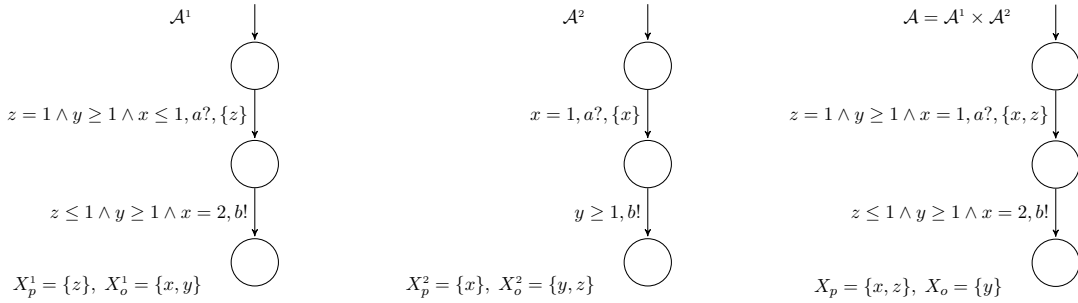


Figure 3: Example of a product $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2$.

Contrary to the parallel product, the set of traces of the product of two OTAIOS is not the intersection of the sets of traces of these TAIOs, as illustrated by the following example.

Example 1.6. Figure 3 artificially illustrates the notion of product of two OTAIOS. One can see that $1.a?.1.b!$ is a trace of \mathcal{A}^1 and \mathcal{A}^2 but is not a trace of $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2$. Indeed, in \mathcal{A}^1 , $1.a?.1.b!$ is the trace of a sequence where x is not reset at the first action. Unfortunately, the clock x is observed by \mathcal{A}^1 but is a proper clock of \mathcal{A}^2 which resets it at the first action. As a consequence, $1.a?.1.b!$ cannot be a trace of the product $\mathcal{A}^1 \times \mathcal{A}^2$. In fact, the second edge in \mathcal{A} can never be fired, since clocks z and x agree on their values and cannot be simultaneously smaller than 1 and equal to 2.

On the other hand, sequences are more adapted to express the underlying operation. To compare the sets of sequences of $\mathcal{A}^1 \times \mathcal{A}^2$ with the sets of sequences of its factors, we introduce an operation that lifts the sets of clocks of factors to the set of clocks of the product: for \mathcal{A}^1 defined on (X_p^1, X_o^1) , and $X_p^1 \cap X_p^2 = \emptyset$, $\mathcal{A}^1 \uparrow^{(X_p^2, X_o^2)}$ denotes an automaton identical to \mathcal{A}^1 but defined on $(X_p^1, X_p^2 \cup X_o^1 \cup X_o^2 \setminus X_p^1)$. The effect on the semantics is to duplicate moves of \mathcal{A}^1 with unconstrained resets in $(X_p^2 \cup X_o^2) \setminus (X_p^1 \cup X_o^1)$, so that $\mathcal{A}^1 \uparrow^{(X_p^2, X_o^2)}$ strongly

³Synchronizing internal actions allows for more precision in test selection. This justifies to have a set of internal actions in the TAIO model.

bisimulates \mathcal{A}^1 . The equivalence just consists in ignoring values of added clocks which do not interfere in the guards. Similarly $\mathcal{A}^2 \uparrow^{(X_p^1, X_o^1)}$ is defined on $(X_p^2, X_p^1 \cup X_o^2 \cup X_o^1 \setminus X_p^2)$. Both $\mathcal{A}^1 \uparrow^{X_p^2, X_o^2}$ and $\mathcal{A}^2 \uparrow^{X_p^1, X_o^1}$ have sequences in $(\mathbb{R}_{\geq 0} \sqcup (\Sigma_\tau^A \times (X_p^1 \cup X_p^2 \cup X_o^1 \cup X_o^2)))^*$. They synchronize on both delays and common actions with their resets. The effect of the product is to restrict the respective environments (observed clocks) by imposing the resets of the peer TAIIO. The sequences of the product are then characterized by

$$\text{Seq}(\mathcal{A}^1 \times \mathcal{A}^2) = \text{Seq}(\mathcal{A}^1 \uparrow^{(X_p^2, X_o^2)}) \cap \text{Seq}(\mathcal{A}^2 \uparrow^{(X_p^1, X_o^1)}) \quad (1.2)$$

meaning that the product of OTAIIOs is the adequate operation for intersecting sets of sequences.

An OTAIIO equipped with a set of states $F \subseteq S^A$ can play the role of an acceptor. A run is *accepted* in F if it ends in F . $\text{Seq}_F(\mathcal{A})$ denotes the set of sequences of accepted runs and $\text{Traces}_F(\mathcal{A})$ the set of their traces. By abuse of notation, if L is a subset of locations in L^A , we note $\text{Seq}_L(\mathcal{A})$ for $\text{Seq}_{L \times \mathbb{R}_{\geq 0}^A}(\mathcal{A})$ and similarly for $\text{Traces}_L(\mathcal{A})$. Note that for the product $\mathcal{A}^1 \times \mathcal{A}^2$, if F^1 and F^2 are subsets of states of \mathcal{A}^1 and \mathcal{A}^2 respectively, additionally to (1.2), the following equality holds:

$$\text{Seq}_{F^1 \times F^2}(\mathcal{A}^1 \times \mathcal{A}^2) = \text{Seq}_{F^1}(\mathcal{A}^1 \uparrow^{(X_p^2, X_o^2)}) \cap \text{Seq}_{F^2}(\mathcal{A}^2 \uparrow^{(X_p^1, X_o^1)}). \quad (1.3)$$

2. CONFORMANCE TESTING THEORY

In this section, we recall the conformance theory for timed automata based on the conformance relation **tioco** [18] that formally defines the set of correct implementations of a given TAIIO specification. **tioco** is a natural extension of the **ioco** relation of Tretmans [23] to timed systems. We then define test cases, formalize their executions, verdicts and expected properties relating verdicts to conformance. Finally, we introduce a refinement relation between TAIIOs that preserves **tioco**, and will be useful in proving test case properties.

2.1. The tioco conformance theory. We consider that the specification is given as a (possibly non-deterministic) TAIIO \mathcal{A} . The implementation is a black box, unknown except for its alphabet of observable actions, which is the same as the one of \mathcal{A} . As usual, in order to formally reason about conformance, we assume that the implementation can be modeled by an (unknown) TAIIO. Formally:

Definition 2.1 (Implementation). Let $\mathcal{A} = (L^A, \ell_0^A, \Sigma_\tau^A, \Sigma_\dagger^A, \Sigma_\tau^A, X_p^A, \emptyset, M^A, I^A, E^A)$ be a specification TAIIO. An implementation of \mathcal{A} is an input-complete and non-blocking TAIIO $\mathcal{I} = (L^{\mathcal{I}}, \ell_0^{\mathcal{I}}, \Sigma_\tau^{\mathcal{I}}, \Sigma_\dagger^{\mathcal{I}}, \Sigma_\tau^{\mathcal{I}}, X_p^{\mathcal{I}}, \emptyset, M^{\mathcal{I}}, I^{\mathcal{I}}, E^{\mathcal{I}})$ with same observable alphabet as \mathcal{A} ($\Sigma_\tau^{\mathcal{I}} = \Sigma_\tau^A$ and $\Sigma_\dagger^{\mathcal{I}} = \Sigma_\dagger^A$). $\mathcal{I}(\mathcal{A})$ denotes the set of possible implementations of \mathcal{A} .

The requirements that an implementation is input-complete and non-blocking will ensure that the execution of a test case on \mathcal{I} does not block before verdicts are emitted.

Among the possible implementations in $\mathcal{I}(\mathcal{A})$, the conformance relation **tioco** (for *timed input-output conformance*) [18] formally defines which ones conform to \mathcal{A} , naturally extending the classical **ioco** relation of Tretmans [23] to timed systems:

Definition 2.2 (Conformance relation). Let \mathcal{A} be a TAIIO representing the specification and $\mathcal{I} \in \mathcal{I}(\mathcal{A})$ be an implementation of \mathcal{A} . We say that \mathcal{I} conforms to \mathcal{A} and write \mathcal{I} **tioco** \mathcal{A} if $\forall \sigma \in \text{Traces}(\mathcal{A}), \text{out}(\mathcal{I} \text{ after } \sigma) \subseteq \text{out}(\mathcal{A} \text{ after } \sigma)$.

Note that **tioco** is equivalent to the **rtioco** relation that was defined independently in [19] (see [22]). Intuitively, \mathcal{I} conforms to \mathcal{A} if after any timed trace enabled in \mathcal{A} , every output or delay of \mathcal{I} is specified in \mathcal{A} . This means that \mathcal{I} may accept more inputs than \mathcal{A} , but is authorized to send less outputs, or send them during a more restricted time interval. The intuition is illustrated on the following simple example:

Example 2.3. Figure 4 represents a specification \mathcal{A} and two possible implementations \mathcal{I}_1 and \mathcal{I}_2 . Note that \mathcal{I}_1 and \mathcal{I}_2 should be input-complete, but for simplicity of figures, we omit some inputs and consider that missing inputs loop to the current location. It is easy to see that \mathcal{I}_1 conforms to \mathcal{A} . Indeed, it accepts more inputs, which is allowed (after the trace ϵ , \mathcal{I}_1 can receive a and d while \mathcal{A} only accepts a), and emits the output b during a more restricted interval of time ($out(\mathcal{I}_1 \text{ after } a.2) = [0, \infty)$ is included in $out(\mathcal{A} \text{ after } a.2) = [0, \infty) \sqcup \{b\}$). On the other hand \mathcal{I}_2 does not conform to \mathcal{A} for two reasons: \mathcal{I}_2 may send a new output c and may send b during a larger time interval (e.g. $out(\mathcal{I}_2 \text{ after } a.1) = [0, \infty) \sqcup \{b, c\}$ is not included in $out(\mathcal{A} \text{ after } a.1) = [0, \infty)$).

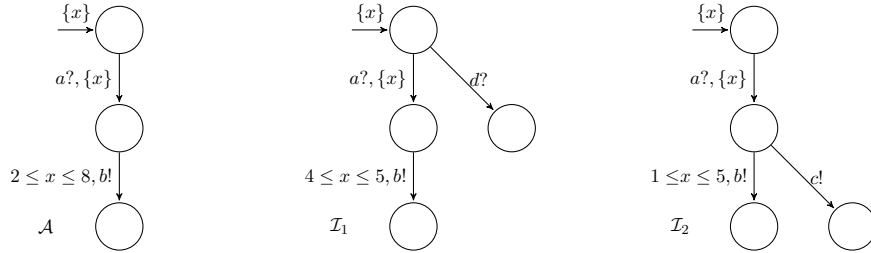


Figure 4: Example of a specification \mathcal{A} and two implementations \mathcal{I}_1 and \mathcal{I}_2 .

In practice, conformance is checked by test cases run on implementations. In our setting, we define test cases as deterministic TAIOS equipped with verdicts defined by a partition of states.

Definition 2.4 (Test suite, test case). Given a specification TAIOS \mathcal{A} , a *test suite* is a set of *test cases*, where a *test case* is a pair $(\mathcal{TC}, \mathbf{Verdicts})$ consisting of:

- a deterministic TAIOS $\mathcal{TC} = (L^{\mathcal{TC}}, \ell_0^{\mathcal{TC}}, \Sigma_?^{\mathcal{TC}}, \Sigma_!^{\mathcal{TC}}, \emptyset, X_p^{\mathcal{TC}}, \emptyset, M^{\mathcal{TC}}, I^{\mathcal{TC}}, E^{\mathcal{TC}})$,
- a partition $\mathbf{Verdicts}$ of the set of states $S^{\mathcal{TC}} = \mathbf{None} \sqcup \mathbf{Inconc} \sqcup \mathbf{Pass} \sqcup \mathbf{Fail}$. States outside \mathbf{None} are called *verdict states*.

We also require that

- $\Sigma_?^{\mathcal{TC}} = \Sigma_!^{\mathcal{A}}$ and $\Sigma_!^{\mathcal{TC}} = \Sigma_?^{\mathcal{A}}$,
- \mathcal{TC} is non-blocking, (e.g. $I^{\mathcal{TC}}(\ell) = \mathbf{true}$ for all $\ell \in L^{\mathcal{TC}}$),
- \mathcal{TC} is input-complete in all \mathbf{None} states, meaning that it is ready to receive any input from the implementation before reaching a verdict.

In the following, for simplicity we will sometimes abuse notations and write \mathcal{TC} instead of $(\mathcal{TC}, \mathbf{Verdicts})$. Let us give some intuition about the different verdicts of test cases. **Fail** states are those where the test case rejects an implementation. The intention is thus to detect a non-conformance. **Pass** and **Inconc** states are linked to test purposes (see Section 4): the intention is that **Pass** states should be those where no non-conformance has been detected and the test purpose is satisfied, whereas **Inconc** states should be those

states where no non-conformance has been detected, but the test purpose cannot be satisfied anymore. **None** states are all other states. We insist on the fact that those are intentional characterizations of the verdicts. Properties of test cases defined later specify whether these intentions are satisfied by test cases. We will see that it is not always the case for all properties.

The execution of a test case $\mathcal{TC} \in \text{Test}(\mathcal{A})$ on an implementation $\mathcal{I} \in \mathcal{I}(\mathcal{A})$ is modeled by the parallel product $\mathcal{I} \parallel \mathcal{TC}$, which entails that $\text{Traces}(\mathcal{I} \parallel \mathcal{TC}) = \text{Traces}(\mathcal{I}) \cap \text{Traces}(\mathcal{TC})$. The facts that \mathcal{TC} is input-complete (in **None** states) and non-blocking while \mathcal{I} is input-complete (in all states) and non-blocking ensure that no deadlock occurs before a verdict is reached.

We say that the verdict of an execution of trace $\sigma \in \text{Traces}(\mathcal{TC})$, noted $\text{Verdict}(\sigma, \mathcal{TC})$, is **Pass**, **Fail**, **Inconc** or **None** if \mathcal{TC} after σ is included in the corresponding states set⁴. We write \mathcal{I} fails \mathcal{TC} if some execution σ of $\mathcal{I} \parallel \mathcal{TC}$ leads \mathcal{TC} to a **Fail** state, *i.e.* when $\text{Traces}_{\text{Fail}}(\mathcal{TC}) \cap \text{Traces}(\mathcal{I}) \neq \emptyset$, which means that there exists $\sigma \in \text{Traces}(\mathcal{I}) \cap \text{Traces}(\mathcal{TC})$ such that $\text{Verdict}(\sigma, \mathcal{TC}) = \text{Fail}$. Notice that this is only a possibility to reach the **Fail** verdict among the infinite set of executions of $\mathcal{I} \parallel \mathcal{TC}$. Hitting one of these executions is not ensured both because of the lack of control of \mathcal{TC} on \mathcal{I} and of timing constraints imposed by these executions.

We now introduce soundness, a crucial property ensured by our test generation method. We also introduce exhaustiveness and strictness that will be ensured when determinization is exact (see Section 4).

Definition 2.5 (Test suite soundness, exhaustiveness and strictness). A test suite \mathcal{TS} for \mathcal{A} is:

- *sound* if $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A}), \forall \mathcal{TC} \in \mathcal{TS}, \mathcal{I} \text{ fails } \mathcal{TC} \Rightarrow \neg(\mathcal{I} \text{ tioco } \mathcal{A})$,
- *exhaustive* if $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A}), \neg(\mathcal{I} \text{ tioco } \mathcal{A}) \Rightarrow \exists \mathcal{TC} \in \mathcal{TS}, \mathcal{I} \text{ fails } \mathcal{TC}$,
- *strict* if $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A}), \forall \mathcal{TC} \in \mathcal{TS}, \neg(\mathcal{I} \parallel \mathcal{TC} \text{ tioco } \mathcal{A}) \Rightarrow \mathcal{I} \text{ fails } \mathcal{TC}$.

Intuitively, soundness means that no conformant implementation can be rejected by the test suite, *i.e.* any failure of a test case during its execution characterizes a non-conformance. Conversely, exhaustiveness means that every non-conformant implementation may be rejected by the test suite. Remember that the definition of \mathcal{I} fails \mathcal{TC} indicates only a possibility of reject. Finally, strictness means that non-conformance is detected once it occurs. In fact, $\neg(\mathcal{I} \parallel \mathcal{TC} \text{ tioco } \mathcal{A})$ means that there is a trace common to \mathcal{TC} and \mathcal{I} which does not conform to \mathcal{A} . The universal quantification on \mathcal{I} and \mathcal{TC} implies that any such trace will fail \mathcal{TC} . In particular, this implies that failure will be detected as soon as it occurs.

Example 2.6. Figure 5 represents a test suite composed of a single test case \mathcal{TC} for the specification \mathcal{A} of the Figure 4. Indeed, \mathcal{TC} is a TAIIO which is input-complete in the **None** states. \mathcal{TS} is sound because the **Fail** states of \mathcal{TC} are reached only when a conformance error occurs, *e.g.* on trace 1.b. However, this test case can observe non-conformant traces without detecting them, hence \mathcal{TS} is not strict. For example, 1.a.1.b, 1.a.1.c and 1.a.9.c are non-conformant traces that do not imply a **Fail** verdict. These traces are *e.g.* traces of \mathcal{I}_2 (Figure 4) which should allow to detect that $\neg(\mathcal{I}_2 \text{ tioco } \mathcal{A})$.

⁴Note that \mathcal{TC} being deterministic, \mathcal{TC} after σ is a singleton.

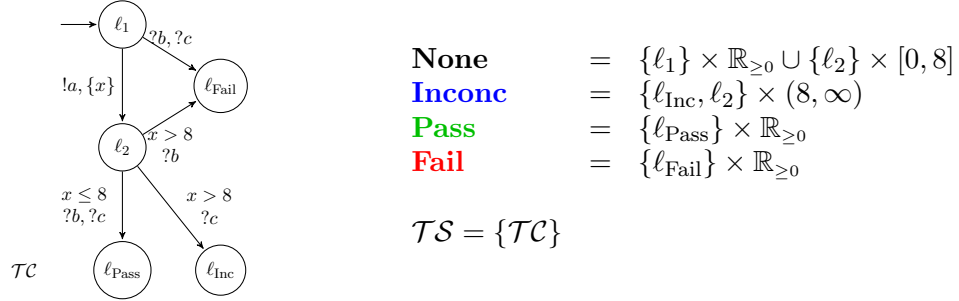


Figure 5: Example of a sound but not strict test suite for the specification \mathcal{A} (Figure 4).

2.2. Refinement preserving tioco. We introduce an io-refinement relation between two TAIOS, a generalization to non-deterministic TAIOS of the io-refinement between DTAIOS introduced in [9], itself a generalization of alternating simulation [2]. Informally \mathcal{A} io-refines \mathcal{B} if \mathcal{A} specifies more inputs and allows less outputs and delays. As a consequence, if \mathcal{A} and \mathcal{B} are specifications, \mathcal{A} is more restrictive than \mathcal{B} with respect to conformance. We thus prove that io-abstraction (the inverse relation) preserves **tioco**: if \mathcal{I} conforms to \mathcal{A} , it also conforms to any io-abstraction \mathcal{B} of \mathcal{A} . This will ensure that soundness of test cases is preserved by the approximate determinization defined in Section 3.

Definition 2.7. Let \mathcal{A} and \mathcal{B} be two TAIOS with same input and output alphabets, we say that \mathcal{A} *io-refines* \mathcal{B} (or \mathcal{B} *io-abstracts* \mathcal{A}) and note $\mathcal{A} \preceq \mathcal{B}$ if

- (i) $\forall \sigma \in \text{Traces}(\mathcal{B}), \text{out}(\mathcal{A} \text{ after } \sigma) \subseteq \text{out}(\mathcal{B} \text{ after } \sigma)$ and,
- (ii) $\forall \sigma \in \text{Traces}(\mathcal{A}), \text{in}(\mathcal{B} \text{ after } \sigma) \subseteq \text{in}(\mathcal{A} \text{ after } \sigma)$.

As we will see below, \preceq is a preorder relation. Moreover, as condition (ii) is always satisfied if \mathcal{A} is input-complete, for $\mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\mathcal{I} \text{ tioco } \mathcal{A}$ is equivalent to $\mathcal{I} \preceq \mathcal{A}$. By transitivity of \preceq , it follows that io-refinement preserves conformance (see Proposition 2.9).

Lemma 2.8. *The io-refinement \preceq is a preorder relation.*

Proof. The relation \preceq is trivially reflexive and we prove that it is transitive.

Suppose that $\mathcal{A} \preceq \mathcal{B}$ and $\mathcal{B} \preceq \mathcal{C}$. By definition of \preceq we have:

$$\forall \sigma \in \text{Traces}(\mathcal{B}), \text{out}(\mathcal{A} \text{ after } \sigma) \subseteq \text{out}(\mathcal{B} \text{ after } \sigma) \quad (1)$$

$$\forall \sigma \in \text{Traces}(\mathcal{A}), \text{in}(\mathcal{B} \text{ after } \sigma) \subseteq \text{in}(\mathcal{A} \text{ after } \sigma) \quad (2) \quad \text{and}$$

$$\forall \sigma \in \text{Traces}(\mathcal{C}), \text{out}(\mathcal{B} \text{ after } \sigma) \subseteq \text{out}(\mathcal{C} \text{ after } \sigma) \quad (3)$$

$$\forall \sigma \in \text{Traces}(\mathcal{B}), \text{in}(\mathcal{C} \text{ after } \sigma) \subseteq \text{in}(\mathcal{B} \text{ after } \sigma) \quad (4)$$

We want to prove that $\mathcal{A} \preceq \mathcal{C}$ thus that

$$\forall \sigma \in \text{Traces}(\mathcal{C}), \text{out}(\mathcal{A} \text{ after } \sigma) \subseteq \text{out}(\mathcal{C} \text{ after } \sigma) \quad (5)$$

$$\forall \sigma \in \text{Traces}(\mathcal{A}), \text{in}(\mathcal{C} \text{ after } \sigma) \subseteq \text{in}(\mathcal{A} \text{ after } \sigma) \quad (6)$$

In order to prove (5), let $\sigma \in \text{Traces}(\mathcal{C})$, and examine the two cases:

- If $\sigma \in \text{Traces}(\mathcal{B}) \cap \text{Traces}(\mathcal{C})$ then (1) and (3) imply $\text{out}(\mathcal{A} \text{ after } \sigma) \subseteq \text{out}(\mathcal{B} \text{ after } \sigma)$ and $\text{out}(\mathcal{B} \text{ after } \sigma) \subseteq \text{out}(\mathcal{C} \text{ after } \sigma)$. Thus $\text{out}(\mathcal{A} \text{ after } \sigma) \subseteq \text{out}(\mathcal{C} \text{ after } \sigma)$ and we are done.

- If $\sigma \in \text{Traces}(\mathcal{C}) \setminus \text{Traces}(\mathcal{B})$, there exist $\sigma', \sigma'' \in (\Sigma_{obs} \sqcup \mathbb{R}_{\geq 0})^*$ and $a \in \Sigma_{obs} \sqcup \mathbb{R}_{\geq 0}$ such that $\sigma = \sigma'.a.\sigma''$ with $\sigma' \in \text{Traces}(\mathcal{B}) \cap \text{Traces}(\mathcal{C})$ and $\sigma'.a \in \text{Traces}(\mathcal{C}) \setminus \text{Traces}(\mathcal{B})$. As $\mathcal{B} \preceq \mathcal{C}$, by (4) we get that $a \in \Sigma_I \sqcup \mathbb{R}_{\geq 0}$. But as $\mathcal{A} \preceq \mathcal{B}$, and $\sigma' \in \text{Traces}(\mathcal{B})$, the condition (1) induces that $\text{out}(\mathcal{A} \text{ after } \sigma') \subseteq \text{out}(\mathcal{B} \text{ after } \sigma')$, and then $\sigma'.a \in \text{Traces}(\mathcal{C}) \setminus \text{Traces}(\mathcal{A})$. We deduce that $\text{out}(\mathcal{A} \text{ after } \sigma'.a) = \emptyset$, and thus $\text{out}(\mathcal{A} \text{ after } \sigma) = \emptyset \subseteq \text{out}(\mathcal{C} \text{ after } \sigma)$.

The proof of (6) is similar. \square

Proposition 2.9. *If $\mathcal{A} \preceq \mathcal{B}$ then $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A}) (= \mathcal{I}(\mathcal{B}))$, $\mathcal{I} \text{ tioco } \mathcal{A} \Rightarrow \mathcal{I} \text{ tioco } \mathcal{B}$.*

Proof. This proposition is a direct consequence of the transitivity of \preceq . In fact when \mathcal{I} is input-complete, by definition $\forall \sigma \in \text{Traces}(\mathcal{I}), \text{in}(\mathcal{I} \text{ after } \sigma) = \Sigma_?$, thus condition (ii) of \preceq trivially holds: $\forall \sigma \in \text{Traces}(\mathcal{I}), \text{in}(\mathcal{A} \text{ after } \sigma) \subseteq \text{in}(\mathcal{I} \text{ after } \sigma)$. Thus $\mathcal{I} \text{ tioco } \mathcal{A}$ (which is defined by $\forall \sigma \in \text{Traces}(\mathcal{A}), \text{out}(\mathcal{I} \text{ after } \sigma) \subseteq \text{out}(\mathcal{A} \text{ after } \sigma)$) is equivalent to $\mathcal{I} \preceq \mathcal{A}$. Now suppose $\mathcal{A} \preceq \mathcal{B}$ and $\mathcal{I} \text{ tioco } \mathcal{A}$ then the transitivity of \preceq gives $\mathcal{I} \text{ tioco } \mathcal{B}$. \square



Figure 6: Counter-example to converse of Proposition 2.9.

Remark: unfortunately, the converse of Proposition 2.9 is in general false, already in the untimed case. This is illustrated in Figure 6. It is clear that the automaton \mathcal{A} accepts all implementations. \mathcal{B} also accepts all implementations as, from the conformance point of view, when a specification does not specify an input after a trace, this is equivalent to specifying this input and then to accept the universal language on Σ_{obs} . Thus $\mathcal{I} \text{ tioco } \mathcal{A} \Rightarrow \mathcal{I} \text{ tioco } \mathcal{B}$. However $\neg(\mathcal{A} \preceq \mathcal{B})$ as $\text{in}(\mathcal{B} \text{ after } \epsilon) = \{a\}$ but $\text{in}(\mathcal{A} \text{ after } \epsilon) = \emptyset$. Notice that this example also works for the untimed case in the **tioco** conformance theory.

As a corollary of Proposition 2.9, we get that io-refinement preserves soundness of test suites:

Corollary 2.10. *If $\mathcal{A} \preceq \mathcal{B}$ then any sound test suite for \mathcal{B} is also sound for \mathcal{A} .*

Proof. Let \mathcal{TS} be a sound test suite for \mathcal{B} . By definition, for any $\mathcal{I} \in \mathcal{I}(\mathcal{B})$, for any $\mathcal{TC} \in \mathcal{TS}$, $\mathcal{I} \text{ fails } \mathcal{TC} \Rightarrow \neg(\mathcal{I} \text{ tioco } \mathcal{B})$. As we have $\mathcal{A} \preceq \mathcal{B}$, by Proposition 2.9, we obtain $\neg(\mathcal{I} \text{ tioco } \mathcal{B}) \Rightarrow \neg(\mathcal{I} \text{ tioco } \mathcal{A})$ which implies that for any $\mathcal{I} \in \mathcal{I}(\mathcal{B})$, for any $\mathcal{TC} \in \mathcal{TS}$, $\mathcal{I} \text{ fails } \mathcal{TC} \Rightarrow \neg(\mathcal{I} \text{ tioco } \mathcal{A})$. Thus \mathcal{TS} is also sound for \mathcal{A} . \square

In the sequel, this corollary will justify our methodology: from \mathcal{A} a non-deterministic TAI, build a deterministic io-abstraction \mathcal{B} of \mathcal{A} , then any test case generated from \mathcal{B} and sound is also sound for \mathcal{A} .

3. APPROXIMATE DETERMINIZATION PRESERVING CONFORMANCE

We recently proposed a game approach to determinize or provide a deterministic over-approximation for TAs [7]. Determinization is exact on all known classes of determinizable TAs (*e.g.* event-clock TAs, TAs with integer resets, strongly non-Zeno TAs) if resources (number and clocks and maximum constant) are sufficient. This method can be adapted to

the context of testing for building a deterministic io-abstraction of a given TAI. Thanks to Proposition 2.9, the construction preserves **tioco**.

The approximate determinization uses the classical region⁵ construction [1]. As for classical timed automata, the regions form a partition of valuations over a given set of clocks which allows to make abstractions in order to decide properties such as the reachability of a location. We note $\text{Reg}_{(X,M)}$ the set of regions over clocks X with maximal constant M . A region r' is a *time-successor* of a region r if $\exists v \in r, \exists t \in \mathbb{R}_{\geq 0}, v + t \in r'$. Given X a set of clocks, a relation over X is a finite conjunction C of atomic constraints of the form $x - y \sim c$ where $x, y \in X$, $\sim \in \{<, =, >\}$ and $c \in \mathbb{N}$. When all constants c belong to $[-M, M]$ for some constant $M \in \mathbb{N}$ we denote by $\text{Rel}_M(X)$ for the set of relations over X . Given a region r , we write $\xrightarrow{r^M}$ for the smallest relation in $\text{Rel}_M(X)$ containing r .

3.1. A game approach to determinize timed automata. The technique presented in [7] applies first to TAs, *i.e.* the alphabet only consists of one kind of actions (say output actions), and the invariants are all trivial. Given such a TA \mathcal{A} over set of clocks $X^{\mathcal{A}}$, a deterministic TA \mathcal{B} with a new set of clocks $X^{\mathcal{B}}$ is built, with $\text{Traces}(\mathcal{A}) = \text{Traces}(\mathcal{B})$ as often as possible, or $\text{Traces}(\mathcal{A}) \subseteq \text{Traces}(\mathcal{B})$. Resources of \mathcal{B} are fixed, and the goal is to simulate the clocks of \mathcal{A} by choosing the right resets in \mathcal{B} . To this aim, letting $k = |X^{\mathcal{B}}|$, a finite 2-player zero-sum turn-based safety game $\mathcal{G}_{\mathcal{A},(k,M^{\mathcal{B}})} = (\mathbf{V}_S, \mathbf{V}_D, \mathbf{v}_0, \delta_S \sqcup \delta_D, \text{Bad})$ is built. The two players, Spoiler and Determinizator, alternate moves, the objective of player Determinizator being to remain in a set of safe states where intuitively, for sure no over-approximation has been performed. In this game, every strategy for Determinizator yields a deterministic automaton \mathcal{B} with $\text{Traces}(\mathcal{A}) \subseteq \text{Traces}(\mathcal{B})$, and every winning strategy induces a deterministic TA \mathcal{B} equivalent to \mathcal{A} . It is well known that for safety games, winning strategies can be chosen positional (*i.e.*, only based on the current state) and computed in linear time in the size of the arena (see *e.g.* [20]).

The game $\mathcal{G}_{\mathcal{A},(k,M^{\mathcal{B}})} = (\mathbf{V}_S, \mathbf{V}_D, \mathbf{v}_0, \delta_S \sqcup \delta_D, \text{Bad})$ is defined as follows:

- $\mathbf{V}_S = 2^{L^{\mathcal{A}} \times \text{Rel}_{\max(M^{\mathcal{A}}, M^{\mathcal{B}})}(X^{\mathcal{A}} \sqcup X^{\mathcal{B}}) \times \{\perp, \top\}} \times \text{Reg}_{(X^{\mathcal{B}}, M^{\mathcal{B}})}$ is the set of states of Spoiler. Each state is a pair $\mathbf{v}_S = (\mathcal{E}, r)$ where r is a region over $X^{\mathcal{B}}$, and \mathcal{E} is a finite set of *configurations* of the form (ℓ, C, b) where ℓ is a location of \mathcal{A} , C is a relation over $X^{\mathcal{A}} \sqcup X^{\mathcal{B}}$ with respect to the maximal constant $M = \max(M^{\mathcal{A}}, M^{\mathcal{B}})$, and b is a boolean marker (\top or \perp). A state of Spoiler thus constitutes a state estimate of \mathcal{A} , and the role of the marker b is to indicate whether over-approximations possibly happened.
- $\mathbf{V}_D = \mathbf{V}_S \times (\Sigma \times \text{Reg}_{(X^{\mathcal{B}}, M^{\mathcal{B}})})$ is the set of states of Determinizator. Each state $\mathbf{v}_D = (\mathbf{v}_S, (a, r'))$ consists of a state of Spoiler, together with an action and a region over $X^{\mathcal{B}}$ which role is to remember the last move of Spoiler.
- $\mathbf{v}_0 = (\{(\ell_0, C_0, b_0)\}, \{\bar{0}\}) \in \mathbf{V}_S$, the initial state of the game, is a state of Spoiler consisting of a single configuration with the initial location ℓ_0 of \mathcal{A} , the simple relation C_0 over $X^{\mathcal{A}} \sqcup X^{\mathcal{B}}$: $\forall x, y \in X^{\mathcal{A}} \sqcup X^{\mathcal{B}}, x - y = 0$, a marker $b_0 = \top$ (no over-approximation was done so far), together with the null region over $X^{\mathcal{B}}$.
- $\delta_S \subseteq \mathbf{V}_S \times (\Sigma \times \text{Reg}_{(X^{\mathcal{B}}, M^{\mathcal{B}})}) \times \mathbf{V}_D$ and $\delta_D \subseteq \mathbf{V}_D \times 2^{X^{\mathcal{B}}} \times \mathbf{V}_S$ are inductively defined from \mathbf{v}_0 as follows:
 - moves of Spoiler are pairs (a, r') and the successor of a state $\mathbf{v}_S = (\mathcal{E}, r) \in \mathbf{V}_S$ by the move (a, r') is simply $\mathbf{v}_D = ((\mathcal{E}, r), (a, r'))$, *i.e.* a copy of \mathbf{v}_S together with a challenge for

⁵Note that it could be adapted to zones with some loss in precision.

Determinizator consisting in an action a and a region $r' \in \text{Reg}_{(X^B, M^B)}$, a time-successor of r ;

- moves of Determinizator are resets $Y \subseteq X^B$ and the successor of a state $\nu_D = ((\mathcal{E}, r), (a, r')) \in \mathbf{V}_D$ by the reset $Y \subseteq X^B$, is the state of Spoiler $(\mathcal{E}', r'_{[Y \leftarrow 0]}) \in \mathbf{V}_S$ where $\mathcal{E}' = \{\text{Succ}_e[(a, r'), Y](\ell, C, b) \mid (\ell, C, b) \in \mathcal{E}\}$ and

$$\text{Succ}_e[(a, r'), Y](\ell, C, b) = \left\{ (\ell', C', b') \left| \begin{array}{l} \exists \ell \xrightarrow{g, a, X} \ell' \in E \text{ s.t. } [r' \cap C]_{|X^A} \cap g \neq \emptyset \\ C' = \overleftarrow{(r' \cap C \cap g)}_{[X \leftarrow 0][Y \leftarrow 0]}^M \\ b' = b \wedge ([r' \cap C]_{|X^A} \subseteq g) \end{array} \right. \right\}.$$

In words, \mathcal{E}' is the set of elementary successors of configurations in \mathcal{E} by (a, r') and by resetting Y . An elementary successor of a configuration (ℓ, C, b) by a transition $\ell \xrightarrow{g, a, X} \ell'$ exists only if the guard $[r' \cap C]_{|X^A}$ over X^A induced by the guard r' over X^B through the relation C intersects g . Intuitively, the transition is possible in ℓ according to the state estimate (ℓ, C) and the region r' . The resulting configuration (ℓ', C', b') is such that:

- * ℓ' is the location reached by the transition;
- * C' is the relation between clocks in X^A and X^B after the moves of the two players, that is after satisfying the guard g in $r' \cap C$, resetting $X \subseteq X^A$ and $Y \subseteq X^B$;
- * b' is a boolean set to \top if both $b = \top$ and the induced guard $[r' \cap C]_{|X^A}$ over X^A implies g . Intuitively, b' becomes \perp when r' encodes more values than g , thus an over-approximation possibly happens.

Note that during the construction of δ_S and δ_D , the states of Determinizator whose successors by δ_D have an empty set of configurations are removed, together with the moves in δ_S leading to them. Indeed these moves have no counterpart in \mathcal{A} .

- **Bad** = $\{(\mathcal{E}, r) \in \mathbf{V}_S \mid \forall (\ell, C, b) \in \mathcal{E}, b = \perp\}$. Bad states Determinizator wants to avoid are states where all configurations are marked \perp , *i.e.* configurations where an approximation possibly happened. Note that a single configuration marked \top in a state is enough to ensure that no over-approximation happened. Indeed, for any path in the game leading to such a state, starting from a \top -marked configuration, and taking elementary predecessors, one can build backwards a sequence of configurations following this path. By definition of the marker's update, these configurations are all marked \top , and the sequence thus corresponds to real traces in the non-deterministic automaton.

Example 3.1. Figure 7 represents a simple non-deterministic timed automaton \mathcal{A} . Let us

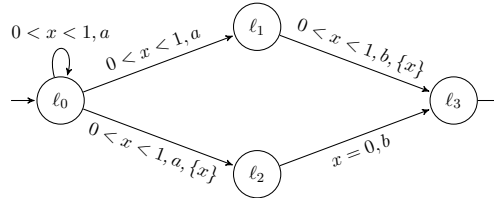
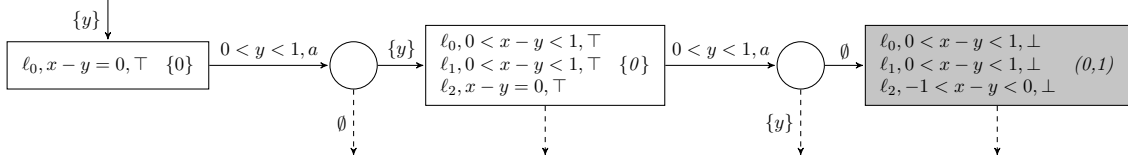


Figure 7: Non-deterministic timed automaton \mathcal{A} .

explain how to construct the game $\mathcal{G}_{\mathcal{A}, (1, 1)}$ for \mathcal{A} with resources $(1, 1)$, that is a single clock y and maximal constant 1. We only detail part of the construction in Figure 8, but the complete game can be found in [7].


 Figure 8: Part of the game $\mathcal{G}_{\mathcal{A},(1,1)}$.

As defined above, the initial state of the game is simply $v_0 = (\{(\ell_0, x - y = 0, \top)\}, \{0\})$.

From v_0 , the only move of Spoiler compatible with behaviors of \mathcal{A} is $0 < y < 1, a$. Corresponding transitions in \mathcal{A} lead to locations ℓ_0 , ℓ_1 and ℓ_2 , and only in this last location x has been reset. Each transition of \mathcal{A} yields a configuration in the next state of Spoiler, and assuming Determinizator chooses to reset y , the three different configurations are the following:

- one with location ℓ_0 , where $x \in (0, 1)$ (no reset in \mathcal{A}) and $y = 0$ (reset in $\mathcal{G}_{\mathcal{A},(1,1)}$),
- one with location ℓ_1 , where $x \in (0, 1)$ and $y = 0$,
- and one with location ℓ_2 , where $x = 0$ (reset in \mathcal{A}) and $y = 0$.

In the two first configurations, the new relation is $\overleftarrow{(y = 0 < x < 1)}^1$, that is $0 < x - y < 1$, and in the last configuration, the new relation is simply $x - y = 0$. As a consequence the successor state is $v_1 = (\{(\ell_0, 0 < x - y < 1, \top), (\ell_1, 0 < x - y < 1, \top), (\ell_2, x - y = 0, \top)\}, \{0\})$. Note that all markers are \top since the guard on y faithfully represented the ones on x .

From state v_1 , if Spoiler chooses the move $0 < y < 1, a$, it is not obvious to which transitions in \mathcal{A} this corresponds, and we thus explain in details how to compute the successor state. First observe that the only configuration in v_1 from which an a action is possible is the first one, with location ℓ_0 . In this configuration, the relation is $0 < x - y < 1$. Let us now explain what guard over x is induced by the relation $C = 0 < x - y < 1$ and the region $r' = 0 < y < 1$. Figure 9 illustrates this computation. The dotted area rep-

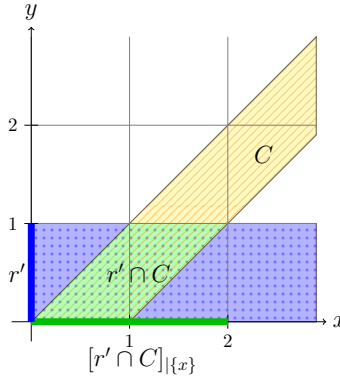


Figure 9: Construction of the induced guard.

resents the set of the valuations over $\{x, y\}$ satisfying the guard $r' = 0 < y < 1$ and the dashed area represents the relation $C = 0 < x - y < 1$. The induced guard $[r' \cap C]_{\{x\}}$ (*i.e.* the guard over x encoded by the guard r' on y through the relation C) is then the

projection over clock x of the intersection of these two areas. In this example, the induced guard is $0 < x < 2$. Therefore, the transitions of \mathcal{A} corresponding to the choice of Spoiler $0 < y < 1, a$ are as before the three ones originating in ℓ_0 , but this time they are over-approximated. Indeed, the induced guard $[r' \cap C]_{\{x\}}$ is not included in the original guard $0 < x < 1$ in \mathcal{A} , *i.e.* *a priori* r' encodes more values than g . As a consequence, all the configurations in Spoiler's successor state are marked \perp . Last, let us detail how the new relations are computed. Assuming Determinizator chooses not to reset y leads to state v_2 , in which for the configuration with location ℓ_0 , the relation is the smallest one containing $(0 < x - y < 1) \cap (0 < y < 1) \cap (0 < x < 1)$, namely $0 < x - y < 1$. The relation for the last configuration in v_2 is $\overleftarrow{((0 < x - y < 1) \cap (0 < y < 1) \cap (0 < x < 1))}_{[x \leftarrow 0]}^1$, which is same as $\overleftarrow{x = 0 < y < 1}^1$, namely $-1 < x - y < 0$.

As explained earlier, a strategy for Determinizator chooses in each state of V_D a set $Y \subseteq X^{\mathcal{B}}$ of clocks to reset. With every strategy Π for Determinizator we associate the TA $\mathcal{B} = \text{Aut}(\Pi)$ obtained by merging a transition of Spoiler with the transition chosen by Determinizator just after. The following theorem links strategies of Determinizator with deterministic over-approximations of the original traces language and enlightens the interest of the game:

Theorem 3.2 ([7]). *Let \mathcal{A} be a TA, and $k, M^{\mathcal{B}} \in \mathbb{N}$. For any strategy Π of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M^{\mathcal{B}})}$, $\mathcal{B} = \text{Aut}(\Pi)$ is a deterministic timed automaton over resources $(k, M^{\mathcal{B}})$ and satisfies $\text{Traces}(\mathcal{A}) \subseteq \text{Traces}(\mathcal{B})$. Moreover, if Π is winning, then $\text{Traces}(\mathcal{A}) = \text{Traces}(\mathcal{B})$.*

When there is no winning strategy, one can either try to increase resources (number of clocks and/or maximal constant), or try to choose the best losing strategy, which is a concern. Indeed, the language inclusion seems to be a good criterion to compare two losing strategies, but it is not a total ordering. Alternatively, one can use the natural heuristics which tends to lose as late as possible (see [6]). In particular, for a game with k clocks and same maximal constant as the original timed automaton, there is a strategy which ensures not to lose before k moves (of each players): by choosing to reset a new clock at each of its moves, Determinizator ensures to perfectly encode all clocks of the original timed automaton. Other alternatives would be to consider heuristics based on quantitative measures over languages.

3.2. Extensions to TAIOS and adaptation to tioco. In the context of model-based testing, the above-mentioned determinization technique must be adapted to TAIOS, as detailed in [6], and summarized below. The model of TAIOS is an expressive model of timed automata incorporating internal actions and invariants. Moreover, inputs and outputs must be treated differently in order to build from a TAIOS \mathcal{A} a DTAIO \mathcal{B} such that $\mathcal{A} \preceq \mathcal{B}$, and then to preserve **tioco**.

- *Internal actions* are naturally part of the specification model. They cannot be observed during test executions and should thus be removed during determinization. In order to do so, a closure by internal actions is performed for each state during the construction of the game, that is, in each state, all the configurations reachable by internal actions are added to the set of configurations. To this attempt, states of the game have to be extended since internal actions might be enabled from a subset of time-successors of the region associated with the state. Therefore, each configuration is associated with a

proper region which is a time-successor of the initial region of the state. The closure by internal actions is effectively computed the same way as successors in the original construction when Determinizator is not allowed to reset any clock. It is well known that timed automata with silent transitions are strictly more expressive than standard timed automata [4]. Therefore, our approximation can be coarse, but it performs as well as possible with its available clock information.

- *Invariants* are classically used to model urgency in timed systems. Taking into account urgency of outputs is quite important, indeed without the ability to express it, for instance, any dummy system would conform to all specifications. Ignoring all invariants in the approximation as done in [18] surely yields an io-abstraction: delays (considered as outputs) are over-approximated. In order to be more precise, while preserving the io-abstraction relation \preceq , with each state of the game is associated the most restrictive invariant containing invariants of all the configurations in the state. In the computation of the successors, invariants are treated as guards and their validity is verified at both ends of the transition. A state whose invariant is strictly over-approximated is treated as unsafe in the game.
- Rather than over-approximating a given TAIIO \mathcal{A} , we aim here at building a DTAIO \mathcal{B} *io-abtracting* \mathcal{A} ($\mathcal{A} \preceq \mathcal{B}$). Successors by outputs are over-approximated as in the original game, while successors by inputs must be under-approximated. The over-approximated closure by silent transitions is not suitable to under-approximation. Therefore, states of the game are extended to contain both over-approximated and under-approximated closures. Thus, the unsafe successors by an input (where possibly an over-approximation would occur), are not built.

Example 3.3. Figure 10 represents a non-deterministic timed automaton \mathcal{A}' that has invariants and internal actions. It is a sub-automaton of the timed automaton we use in the next section (see Figure 13) to illustrate the approximate determinization for our test selection.

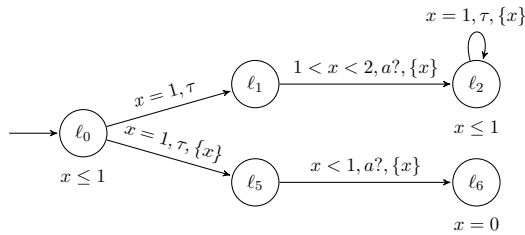
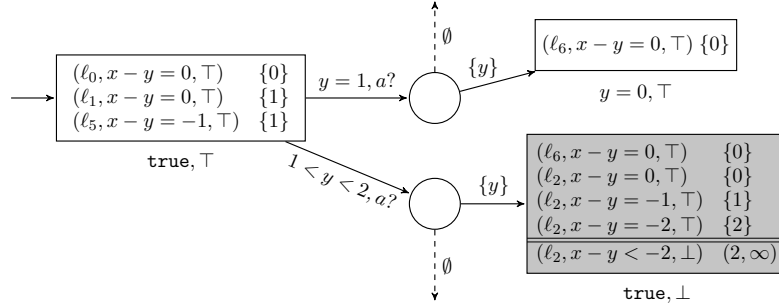


Figure 10: Non-deterministic timed automaton \mathcal{A}' (with invariants and internal actions).

Using this automaton \mathcal{A}' , let us illustrate how the game construction is adapted to deal with internal actions and invariants, by detailing part of the game $\mathcal{G}_{\mathcal{A}',(1,2)}$ represented in Figure 11.

A state of Spoiler in the game is a triple $(S_-, S_+, (I, b_I))$ where S_- (resp. S_+) is the under-approximated (resp. over-approximated) closure by unobservable actions of the successors by some observable action, I is the invariant and b_I is the marker which indicates a risk of approximation of the invariant. The invariant and the marker of Spoiler's states are written below the states.

Figure 11: Part of the game $\mathcal{G}_{\mathcal{A}',(1,2)}$.

In the initial state of the game, $(\ell_0, x-y=0, \top, \{0\}) \in S_- \subseteq S_+$. Moreover, an internal action τ can be fired for $x=1$ along two different edges, which add two configurations, associated with the region $y=1$ (because $x-y=0$ in the first configuration). Determinizator cannot reset y along an internal action, hence the relation for the configuration with location ℓ_5 is $x-y=-1$. Note that the region $y=1$ is associated with the two last configurations in the initial state, reflecting that the internal action fired and thus the least value for y is 1. Also in this case, the closure (by internal actions) is not approximated, hence $S_- = S_+$. On the other hand, it may be surprising that the invariant of this initial state is **true** whereas the invariant of the initial state of \mathcal{A}' is $x \leq 1$. In fact, the invariant of a state is the smallest invariant containing the union, over all its configurations, of induced invariants. On this example, after an internal action from ℓ_0 , delays are not constrained anymore in ℓ_1 and ℓ_6 (invariants are **true**). Thus the invariant in the initial state of the game is not approximated, so its marker is \top .

From this initial state, Spoiler can choose the regions $y=1$ or $1 < y < 2$ together with action $a?$. For $y=1$, this can only happen from the configuration with location ℓ_5 . Indeed, the relation $x-y=0$ and the guard $y=1$ induce a guard $x=1$ which is not compatible with the outgoing edge from ℓ_1 in \mathcal{A}' . The computation of the successor state, *e.g.* when Determinizator chooses to reset y , is simple: no internal action is fireable and the invariant in ℓ_6 is precisely expressed by $y=0$. The situation is more complex when Spoiler chooses the region $1 < y < 2$: in this case there are two successors by the observable action $a?$ (leading to locations ℓ_6 and ℓ_2), and for the first one internal actions may follow. We thus have to compute the closure by internal actions of the successor configuration by observable action $a?$. Before computing the closure, and assuming that Determinizator resets clock y , the successor state is composed of two configurations: $(\ell_2, x-y=0, \top)$ and $(\ell_6, x-y=0, \top)$ together with region $y=0$. Along the τ -loop on location ℓ_2 , x is reset in \mathcal{A}' whereas y cannot be reset in the game (because it is an internal action). Starting from configuration $(\ell_2, x-y=0, \top, \{0\})$ and performing once the internal action τ , the resulting configuration is thus $(\ell_2, x-y=-1, \top, \{1\})$. This computation is iterated to obtain the closure by internal actions, which in such a case, will depend on the maximal constant (here 2). Indeed, after $(\ell_2, x-y=-1, \top, \{1\})$, the next configuration is $(\ell_2, x-y=-2, \top, \{2\})$ and starting from $(\ell_2, x-y=-2, \top, \{2\})$ the effect of one internal action would yield to $(\ell_2, x-y=-3, \top, \{3\})$. However, $x-y=-3$ cannot be expressed in $\text{Rel}_2(\{x, y\})$, so it is approximated by the least relation of $\text{Rel}_2(\{x, y\})$ containing it, that is $x-y < -2$. Similarly, region $y=3$ is approximated by $y > 2$. As a consequence, the configuration

$(\ell_2, x - y = -3, \top, \{3\})$ is approximated by $(\ell_2, x - y < -2, \perp, (2, \infty))$ in S_+ . Note that this latter configuration is in $S_+ \setminus S_-$ and thus separated from configurations in S_- by two horizontal lines on Figure 11. Moreover, taking the union of all the invariants, we obtain **true** as invariant for this state, but since it is approximated for the last configuration $(\ell_2, x - y < -2, \perp, (2, \infty))$, its marker is \perp .

All in all, these modifications allow to deal with the full TAIIO model with invariants, internal transitions and inputs/outputs. In particular, the treatment of invariants is consistent with the io-abstraction: delays are considered as outputs, thus over-approximated. Figure 14 represents a part of this game for the TAIIO of Figure 13. The new game then enjoys the following nice property:

Proposition 3.4 ([6]). *Let \mathcal{A} be a TAIIO, and $k, M^{\mathcal{B}} \in \mathbb{N}$. For any strategy Π of Determinizator in the game $\mathcal{G}_{\mathcal{A},(k,M^{\mathcal{B}})}$, $\mathcal{B} = \text{Aut}(\Pi)$ is a DTAIO over resources $(k, M^{\mathcal{B}})$ with $\mathcal{A} \preceq \mathcal{B}$. Moreover, if Π is winning, then $\text{Traces}(\mathcal{A}) = \text{Traces}(\mathcal{B})$.*

In other words, the approximations produced by our method are deterministic io-abstractions of the initial specification, hence the approximate determinization preserves **tioco** (Proposition 2.9), and conversely, sound test cases of the approximate determinization remain sound for the original specification (Corollary 2.10). Note that the proof of proposition 3.4 in [6] considers a stronger refinement relation, thus implies the same result for the present refinement relation. In comparison with our method, the algorithm proposed in [18] always performs an over-approximation, and thus preserves **tioco** only if the specification is input-complete; moreover all invariants are set to **true** in the resulting automata, so the construction does not preserve urgency.

Complexity. The number of regions (resp. relations) over a set of clocks is exponential in the number of clocks. Thus, the number of possible configurations in the game is at most exponential in the cardinality of $X \sqcup Y$ and linear in the number of locations in \mathcal{A} . As a consequence, the size of the game (*i.e.*, number of states in the arena) is at most doubly exponential in $|X \sqcup Y|$ and exponential in $|L^{\mathcal{A}}|$. In particular this bound also holds for the size of the generated deterministic TAIIO, for every *memoryless* strategy of Determinizator. The overall complexity of this io-abstracting determinization algorithm is thus doubly exponential in the size of the instance (original TAIIO and resources).

4. OFF-LINE TEST CASE GENERATION

In this section, we describe the off-line generation of test cases from timed automata specifications and test purposes. We first define test purposes, their role in test generation and their formalization as OTAIOS. We then detail the process of off-line test selection guided by test purposes, which uses the approximate determinization just defined. We also prove properties of generated test cases with respect to conformance and test purposes.

4.1. Test purposes. In testing practice, especially when test cases are generated manually, each test case has a particular objective, informally described by a sentence called test purpose. In formal test generation, test purposes should be formal models interpreted as means to select behaviors to be tested, either focusing on usual behaviors, or on suspected errors in implementations [13], thus typically reachability properties. They complement other selection mechanisms such as coverage methods [26] which, contrary to test purposes, are most often based on syntactical criteria rather than semantic aspects. Moreover, the set of goals covering a given criterion (*e.g.* states, transitions, etc) may be translated into a set of test purposes, each test purpose focusing on one such goal.

As test purposes are selectors of behaviors, a natural way to formalize them is to use a logical formula characterizing a set of behaviors or an automaton accepting those behaviors. In this work we choose to describe test purposes as OTAIOs equipped with accepting states. The motivation is to use a model close to the specification model, easing the description of targeted specification behaviors. The following definition formalizes test purposes, and some alternatives are discussed in Section 5.

Definition 4.1 (Test purpose). Let $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_{?}^{\mathcal{A}}, \Sigma_{!}^{\mathcal{A}}, \Sigma_{\tau}^{\mathcal{A}}, X_p^{\mathcal{A}}, \emptyset, M^{\mathcal{A}}, I^{\mathcal{A}}, E^{\mathcal{A}})$ be a TAIO specification. A *test purpose* for \mathcal{A} is a pair $(\mathcal{TP}, \text{Accept}^{\mathcal{TP}})$ where:

- $\mathcal{TP} = (L^{\mathcal{TP}}, \ell_0^{\mathcal{TP}}, \Sigma_{?}^{\mathcal{TP}}, \Sigma_{!}^{\mathcal{TP}}, \Sigma_{\tau}^{\mathcal{TP}}, X_o^{\mathcal{TP}}, X_p^{\mathcal{TP}}, M^{\mathcal{TP}}, I^{\mathcal{TP}}, E^{\mathcal{TP}})$ is a complete OTAIO (in particular $I^{\mathcal{TP}}(\ell) = \mathbf{true}$ for any $\ell \in L^{\mathcal{TP}}$) with $X_o^{\mathcal{TP}} = X_p^{\mathcal{A}}$ (\mathcal{TP} observes proper clocks of \mathcal{A}) and $X_p^{\mathcal{TP}} \cap X_p^{\mathcal{A}} = \emptyset$,
- $\text{Accept}^{\mathcal{TP}} \subseteq L^{\mathcal{TP}}$ is a subset of trap locations.

In the following, we will sometimes abuse notations and use \mathcal{TP} instead of the pair $(\mathcal{TP}, \text{Accept}^{\mathcal{TP}})$. During the test generation process, test purposes are synchronized with the specification, and together with their **Accept** locations, they will play the role of acceptors of timed behaviors. They are non-intrusive in order not to constrain behaviors of the specification. This explains why they are complete, thus allowing all actions in all locations, and are not constrained by invariants. They observe behaviors of specifications by synchronizing with their actions (inputs, outputs and internal actions) and their proper clocks (by the definition of the product (Definition 1.5), observed clocks of \mathcal{TP} are proper clocks of \mathcal{A} , which mean that \mathcal{TP} does not reset those clocks). However, in order to add some flexibility in the description of timed behaviors, they may have their own proper clocks.

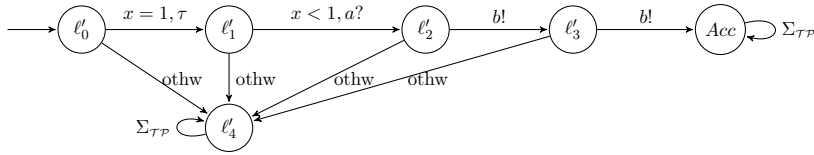


Figure 12: Test purpose \mathcal{TP} .

Example 4.2. Figure 12 represents a test purpose \mathcal{TP} for the specification \mathcal{A} of Figure 1. This one has no proper clock and observes the unique clock x of \mathcal{A} . It accepts sequences where τ occurs at $x = 1$, followed by an input a at $x < 1$ (thus focusing on the lower branch of \mathcal{A} where x is reset), and two subsequent b 's. The label *othw* (for otherwise) on a transition is an abbreviation for the complement of specified transitions leaving the same location. For example in location ℓ'_1 , *othw* stands for $\{(\mathbf{true}, \tau), (\mathbf{true}, b!), (x \geq 1, a?)\}$.

4.2. Principle of test generation. Given a specification TAIIO \mathcal{A} and a test purpose $(\mathcal{TP}, \text{Accept}^{\mathcal{TP}})$, the aim is to build a sound and, if possible strict test case $(\mathcal{TC}, \mathbf{Verdicts})$ focusing on behaviors accepted by \mathcal{TP} . As \mathcal{TP} accepts sequences of \mathcal{A} , but test cases observe timed traces, the intention is that \mathcal{TC} should deliver **Pass** verdicts on traces of sequences of \mathcal{A} accepted by \mathcal{TP} in $\text{Accept}^{\mathcal{TP}}$. This property is formalized by the following definition:

Definition 4.3. A test suite \mathcal{TS} for \mathcal{A} and \mathcal{TP} is said to be *precise* if for any test case \mathcal{TC} in \mathcal{TS} , for any timed observation σ in $\text{Traces}(\mathcal{TC})$, $\mathbf{Verdict}(\sigma, \mathcal{TC}) = \mathbf{Pass}$ if and only if $\sigma \in \text{Traces}(\text{Seq}(\mathcal{A} \uparrow^{(X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}})}) \cap \text{Seq}_{\text{Accept}^{\mathcal{TP}}}(\mathcal{TP}))$.

Let $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_{\tau}^{\mathcal{A}}, \Sigma_{\uparrow}^{\mathcal{A}}, \Sigma_{\tau}^{\mathcal{A}}, X_p^{\mathcal{A}}, \emptyset, M^{\mathcal{A}}, I^{\mathcal{A}}, E^{\mathcal{A}})$ be the specification TAIIO, and $\mathcal{TP} = (L^{\mathcal{TP}}, \ell_0^{\mathcal{TP}}, \Sigma_{\tau}^{\mathcal{TP}}, \Sigma_{\uparrow}^{\mathcal{TP}}, \Sigma_{\tau}^{\mathcal{TP}}, X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}}, M^{\mathcal{TP}}, I^{\mathcal{TP}}, E^{\mathcal{TP}})$ be a test purpose for \mathcal{A} , with its set $\text{Accept}^{\mathcal{TP}}$ of accepting locations. The generation of a test case \mathcal{TC} from \mathcal{A} and \mathcal{TP} proceeds in several steps. First, sequences of \mathcal{A} accepted by \mathcal{TP} are identified by the computation of the product \mathcal{P} of those OTAIOs. Then a determinization step is necessary to characterize conformant traces as well as traces of accepted sequences. Then the resulting deterministic TAIIO \mathcal{DP} is transformed into a test case TAIIO \mathcal{TC}' with verdicts assigned to states. Finally, the test case \mathcal{TC} is obtained by a selection step which tries to avoid some **Inconc** verdicts. The different steps of the test generation process from \mathcal{A} and \mathcal{TP} are detailed in the following paragraphs.

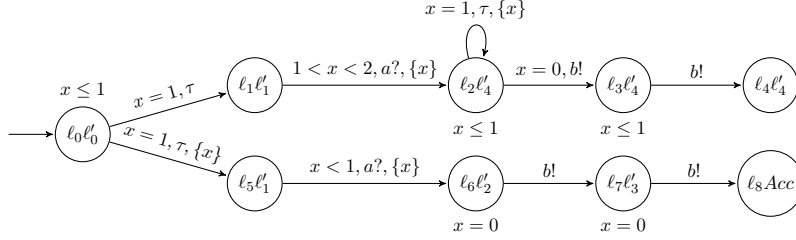
Computation of the product: First, the product $\mathcal{P} = \mathcal{A} \times \mathcal{TP}$ is built (see Definition 1.5 for the definition of the product), associated with the set of marked locations $\text{Accept}^{\mathcal{P}} = L^{\mathcal{A}} \times \text{Accept}^{\mathcal{TP}}$. Let $P = (L^{\mathcal{P}}, \ell_0^{\mathcal{P}}, \Sigma_{\tau}^{\mathcal{P}}, \Sigma_{\uparrow}^{\mathcal{P}}, \Sigma_{\tau}^{\mathcal{P}}, X_p^{\mathcal{P}}, X_o^{\mathcal{P}}, M^{\mathcal{P}}, I^{\mathcal{P}}, E^{\mathcal{P}})$. As $X_o^{\mathcal{TP}} = X_p^{\mathcal{A}}$, we get $X_o^{\mathcal{P}} = \emptyset$ and $X_p^{\mathcal{P}} = X_p^{\mathcal{A}} \sqcup X_p^{\mathcal{TP}}$, thus \mathcal{P} is in fact a TAIIO.

The effect of the product is to unfold \mathcal{A} and to mark locations of the product by $\text{Accept}^{\mathcal{P}}$, so that sequences of \mathcal{A} accepted by \mathcal{TP} are identified. As \mathcal{TP} is complete, $\text{Seq}(\mathcal{TP}) \downarrow_{X_p^{\mathcal{TP}}} = (\mathbb{R}_{\geq 0} \times (\Sigma^{\mathcal{TP}} \times 2^{X_o^{\mathcal{TP}}}))^*$, thus, by the properties of the product (see equation 1.2), $\text{Seq}(\mathcal{P}) \downarrow_{X_p^{\mathcal{TP}}} = \text{Seq}(\mathcal{A})$ *i.e.* the sequences of the product after removing resets of proper clocks of \mathcal{TP} are the sequences of \mathcal{A} . As a consequence $\text{Traces}(\mathcal{P}) = \text{Traces}(\mathcal{A})$, which entails that \mathcal{P} and \mathcal{A} define the same sets of conformant implementations.

Considering accepted sequences of the product \mathcal{P} , by equation 1.3 we get the equality $\text{Seq}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P}) = \text{Seq}(\mathcal{A} \uparrow^{(X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}})}) \cap \text{Seq}_{\text{Accept}^{\mathcal{TP}}}(\mathcal{TP})$, which induces the desired characterization of accepted traces: $\text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P}) = \text{Traces}(\text{Seq}(\mathcal{A} \uparrow^{(X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}})}) \cap \text{Seq}_{\text{Accept}^{\mathcal{TP}}}(\mathcal{TP}))$.

Using the notation $\text{pref}(T)$ for the set of prefixes of traces in a set of traces T , we note $\text{RTraces}(\mathcal{A}, \mathcal{TP}) = \text{Traces}(\mathcal{A}) \setminus \text{pref}(\text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P}))$ for the set of traces of \mathcal{A} which are not prefixes of accepted traces of \mathcal{P} . In the sequel, the principle of test selection will be to try to select traces in $\text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P})$ (and assign to them the **Pass** verdict) and to try to avoid or at least detect (with an **Inconc** verdict) those traces in $\text{RTraces}(\mathcal{A}, \mathcal{TP})$, as these traces cannot be prefixes of traces of sequences satisfying the test purpose.

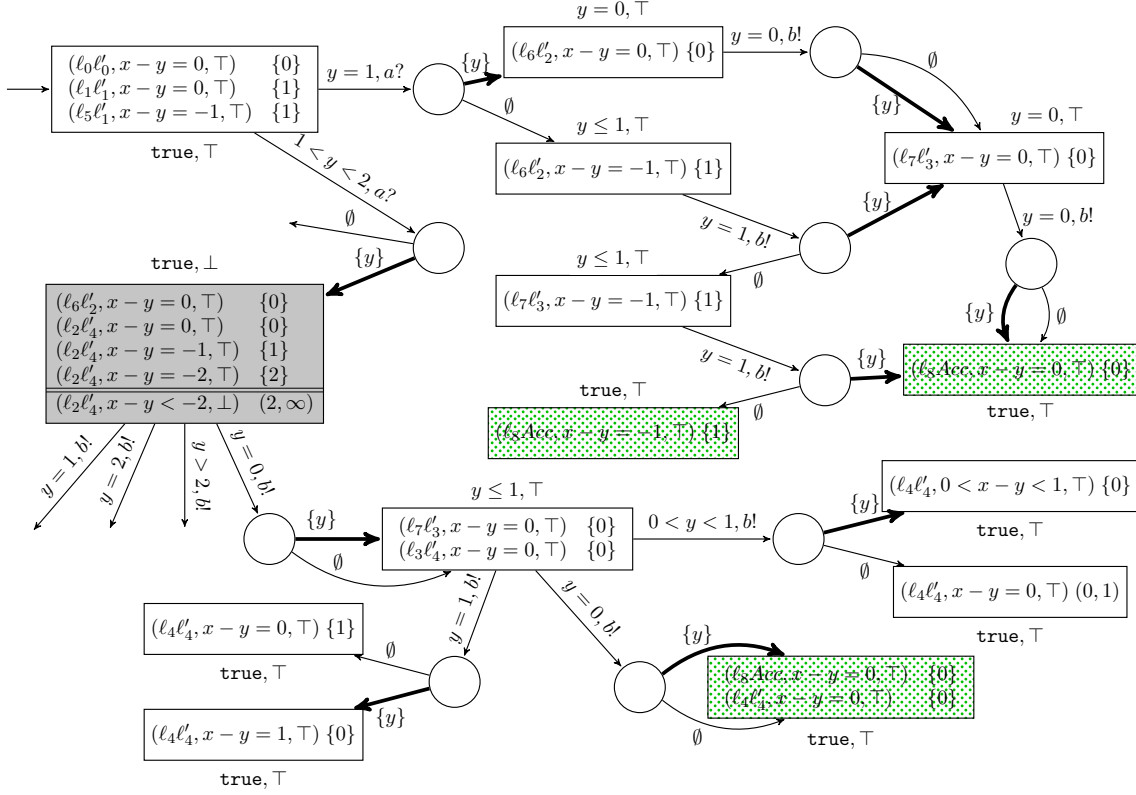
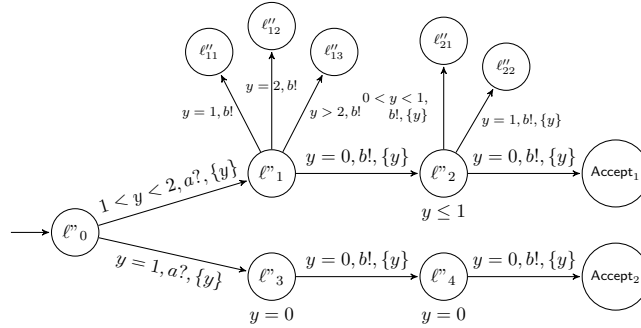
Example 4.4. Figure 13 represents the product \mathcal{P} for the specification \mathcal{A} in Figure 1 and the test purpose \mathcal{TP} in Figure 12. As \mathcal{TP} describes one branch of \mathcal{A} , the product is very simple in this case, *e.g.* intersection of guards are trivial. The only difference with \mathcal{A} is the tagging with $\text{Accept}^{\mathcal{P}}$.

Figure 13: Product $\mathcal{P} = \mathcal{A} \times \mathcal{TP}$.

Approximate determinization of \mathcal{P} into \mathcal{DP} : We now want to transform \mathcal{P} into a deterministic TAIIO \mathcal{DP} such that $\mathcal{P} \preceq \mathcal{DP}$, which by Proposition 2.9) will entail that implementations conformant to \mathcal{P} (thus to \mathcal{A}) are still conformant to \mathcal{DP} . If \mathcal{P} is already deterministic, we simply take $\mathcal{DP} = \mathcal{P}$. Otherwise, the approximate determinization of Section 3 provides a solution. The user fixes some resources $(k, M^{\mathcal{DP}})$, then a deterministic io-abstraction \mathcal{DP} of \mathcal{P} with resources $(k, M^{\mathcal{DP}})$ is computed. By Proposition 3.4, we thus get that \mathcal{DP} io-abstracts \mathcal{P} . \mathcal{DP} is equipped with the set of marked locations $\text{Accept}^{\mathcal{DP}}$ consisting of locations in $L^{\mathcal{DP}}$ containing some configuration whose location is in $\text{Accept}^{\mathcal{P}}$. As a consequence traces of \mathcal{DP} which are traces of sequences accepted by \mathcal{P} in $\text{Accept}^{\mathcal{P}}$ are accepted by \mathcal{DP} in $\text{Accept}^{\mathcal{DP}}$, formally $\text{Traces}(\mathcal{DP}) \cap \text{Traces}(\text{Seq}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P})) = \text{Traces}(\mathcal{DP}) \cap \text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P}) \subseteq \text{Traces}_{\text{Accept}^{\mathcal{DP}}}(\mathcal{DP})$. This means that extra accepted traces may be added due to over-approximations, some traces may be lost (including accepted ones) by under-approximations, but if the under-approximation preserves some traces that are accepted in \mathcal{P} , these are still accepted in \mathcal{DP} . If the determinization is exact (or \mathcal{P} is already deterministic), of course we get more precise relations between the traces and accepted traces of \mathcal{P} and \mathcal{DP} , namely $\text{Traces}(\mathcal{DP}) = \text{Traces}(\mathcal{P})$ and $\text{Traces}_{\text{Accept}^{\mathcal{DP}}}(\mathcal{DP}) = \text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P})$.

Example 4.5. Figure 14 partially represents the game $\mathcal{G}_{\mathcal{P},(1,2)}$ for the TAIIO \mathcal{P} of Figure 13 where, for readability reasons, some behaviors not co-reachable from $\text{Accept}^{\mathcal{DP}}$ (dotted green states) are omitted. Notice that the construction of the initial part of the game was explained in Example 3.3. A strategy Π for Determinizator is represented by bold arrows. Π is not winning (the unsafe configuration, in gray, is unavoidable from the initial state), and in fact an approximation is performed. \mathcal{DP} , represented in Figure 15 is simply obtained from $\mathcal{G}_{\mathcal{P},(1,2)}$ and the strategy Π by merging transitions of Spoiler and those of Determinizator in the strategy.

Generating \mathcal{TC}' from \mathcal{DP} : The next step consists in building a test case $(\mathcal{TC}', \text{Verdicts})$ from \mathcal{DP} . The main point is the computation of verdicts. **Pass** verdicts are simply defined from $\text{Accept}^{\mathcal{DP}}$. **Fail** verdicts that should detect unexpected outputs and delays, rely on a complementation. The difficult part is the computation of **Inconc** states which should detect when $\text{Accept}^{\mathcal{DP}}$ is not reachable (or equivalently **None** states, those states where $\text{Accept}^{\mathcal{DP}}$ is still reachable) and thus relies on an analysis of the co-reachability to locations $\text{Accept}^{\mathcal{DP}}$. Another interesting point is the treatment of invariants. First \mathcal{TC}' will have no invariants (which ensures that it is non-blocking). Second, invariants in \mathcal{DP} are shifted to guards in \mathcal{TC}' and in the definition of **Fail** so that test cases check that the urgency specified in \mathcal{A} is satisfied by \mathcal{I} .


 Figure 14: Game $\mathcal{G}_{\mathcal{P},(1,2)}$.

 Figure 15: Deterministic automaton $\mathcal{DP} = \text{Aut}(\text{II})$.

The test case constructed from $\mathcal{DP} = (L^{\mathcal{DP}}, \ell_0^{\mathcal{DP}}, \Sigma_7^{\mathcal{DP}}, \Sigma_1^{\mathcal{DP}}, \emptyset, X_p^{\mathcal{DP}}, \emptyset, M^{\mathcal{DP}}, I^{\mathcal{DP}}, E^{\mathcal{DP}})$ and $\text{Accept}^{\mathcal{DP}}$ is the pair $(\mathcal{TC}', \text{Verdicts})$ where:

- $\mathcal{TC}' = (L^{\mathcal{TC}'}, \ell_0^{\mathcal{TC}'}, \Sigma_7^{\mathcal{TC}'}, \Sigma_1^{\mathcal{TC}'}, \emptyset, X_p^{\mathcal{TC}'}, \emptyset, M^{\mathcal{TC}'}, I^{\mathcal{TC}'}, E^{\mathcal{TC}'})$ is the TAIIO such that:
 - $L^{\mathcal{TC}'} = L^{\mathcal{DP}} \sqcup \{\ell_{\text{Fail}}\}$ where ℓ_{Fail} is a new location;
 - $\ell_0^{\mathcal{TC}'} = \ell_0^{\mathcal{DP}}$ is the initial location;
 - $\Sigma_7^{\mathcal{TC}'} = \Sigma_1^{\mathcal{DP}} = \Sigma_1^{\mathcal{A}}$ and $\Sigma_1^{\mathcal{TC}'} = \Sigma_7^{\mathcal{DP}} = \Sigma_7^{\mathcal{A}}$, *i.e.* input/output alphabets are mirrored in order to reflect the opposite role of actions in the synchronization of \mathcal{TC}' and \mathcal{I} ;

- $X_p^{\mathcal{T}C'} = X_p^{\mathcal{D}P}$ and $X_o^{\mathcal{T}C'} = X_o^{\mathcal{D}P} = \emptyset$;
- $M^{\mathcal{T}C'} = M^{\mathcal{D}P}$;
- $I^{\mathcal{T}C'}(\ell) = \mathbf{true}$ for any $\ell \in L^{\mathcal{T}C'}$;
- $E^{\mathcal{T}C'} = E_I^{\mathcal{D}P} \sqcup E_{\ell_{\mathbf{Fail}}}$ where

$$E_I^{\mathcal{D}P} = \{(\ell, g \wedge I^{\mathcal{D}P}(\ell), a, X', \ell') \mid (\ell, g, a, X', \ell') \in E^{\mathcal{D}P}\} \text{ and}$$

$$E_{\ell_{\mathbf{Fail}}} = \left\{ (\ell, \bar{g} \wedge I^{\mathcal{D}P}(\ell), a, X_p^{\mathcal{T}C'}, \ell_{\mathbf{Fail}}) \mid \begin{array}{l} \ell \in L^{\mathcal{D}P}, a \in \Sigma_1^{\mathcal{D}P} \\ \text{and } \bar{g} = \neg \bigvee_{(\ell, g, a, X', \ell') \in E^{\mathcal{D}P}} g \end{array} \right\}$$

• **Verdicts** is the partition of $S^{\mathcal{D}P}$ defined as follows:

- **Pass** = $\bigcup_{\ell \in \text{Accept}^{\mathcal{D}P}} (\{\ell\} \times I^{\mathcal{D}P}(\ell))$,
- **None** = $\text{coreach}(\mathcal{D}P, \mathbf{Pass}) \setminus \mathbf{Pass}$,
- **Fail** = $\{\ell_{\mathbf{Fail}}\} \times \mathbb{R}_{\geq 0}^{X^{\mathcal{T}C'}} \sqcup \{(\ell, \neg I^{\mathcal{D}P}(\ell)) \mid \ell \in L^{\mathcal{D}P}\}$;
- **Inconc** = $S^{\mathcal{D}P} \setminus (\mathbf{Pass} \sqcup \mathbf{Fail} \sqcup \mathbf{None})$,

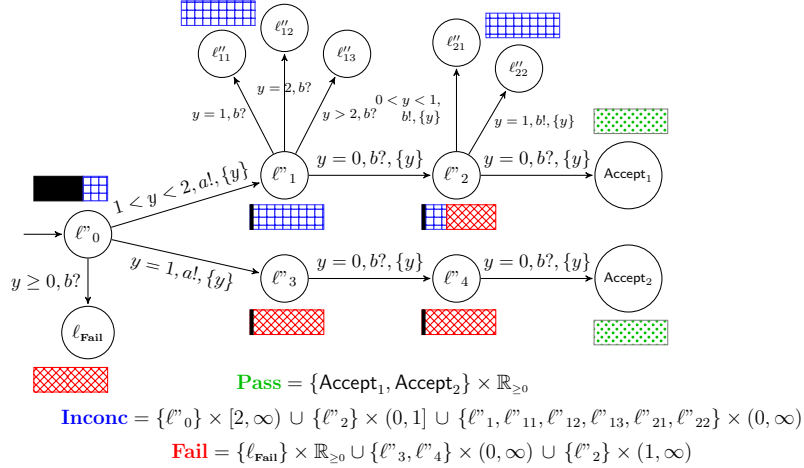
The important points to understand in the construction of $\mathcal{T}C'$ are the completion to **Fail** and the computation of **None**, which, together with **Pass**, define **Inconc** by complementation.

For the completion to **Fail**, the idea is to detect unspecified outputs and delays with respect to $\mathcal{D}P$. Remember that outputs of $\mathcal{D}P$ are inputs of $\mathcal{T}C'$. Moreover, authorized delays in $\mathcal{D}P$ are defined by invariants, but remember that test cases have no invariants (they are **true** in all locations). First, all states in $(\ell, \neg I^{\mathcal{D}P}(\ell))$, $\ell \in L^{\mathcal{D}P}$, *i.e.* states where the invariant runs out, are put into **Fail** which reflects the counterpart in $\mathcal{T}C'$ of the urgency in $\mathcal{D}P$. Then, in each location ℓ , the invariant $I^{\mathcal{D}P}(\ell)$ in $\mathcal{D}P$ is removed and shifted to guards of all transitions leaving ℓ in $\mathcal{T}C'$, as defined in $E_I^{\mathcal{D}P}$. Second, in any location ℓ , for each input $a \in \Sigma_1^{\mathcal{T}C'} = \Sigma_1^{\mathcal{D}P}$, a transition leading to $\ell_{\mathbf{Fail}}$ is added, labeled with a , and whose guard is the conjunction of $I(\ell)$ with the negation of the disjunction of all guards of transitions labeled by a and leaving ℓ (thus **true** if no a -action leaves ℓ), as defined in $E_{\ell_{\mathbf{Fail}}}$. It is then easy to see that $\mathcal{T}C'$ is input-complete in all states.

The computation of **None** is based on an analysis of the co-reachability to **Pass**. **None** contains all states co-reachable from locations in **Pass**. Notice that the set of states $\text{coreach}(\mathcal{D}P, \mathbf{Pass})$, and thus **None**, can be computed symbolically as usual in the region graph of $\mathcal{D}P$, or more efficiently using zones.

Example 4.6. Figure 16 represents the test case $\mathcal{T}C'$ obtained from $\mathcal{D}P$. For readability reasons, we did not represent transitions in $E_{\ell_{\mathbf{Fail}}}$, except the one leaving ℓ''_0 . In fact these are removed in the next selection phase as they are only fireable from states where a verdict has already been issued. The rectangles attached to locations represent the verdicts in these locations when clock y progresses between 0 and 2, and after 2: dotted green for *Pass*, black for **None**, blue grid for **Inconc** and crosshatched red for **Fail**. For example, in ℓ''_2 , the verdict is initially **None**, becomes **Inconc** if no b is received immediately, and even **Fail** if no b is received before one time unit. Notice that in order to reach a **Pass** verdict, one should initially send a after one and strictly before two time units, and expect to receive two consecutive b 's immediately after.

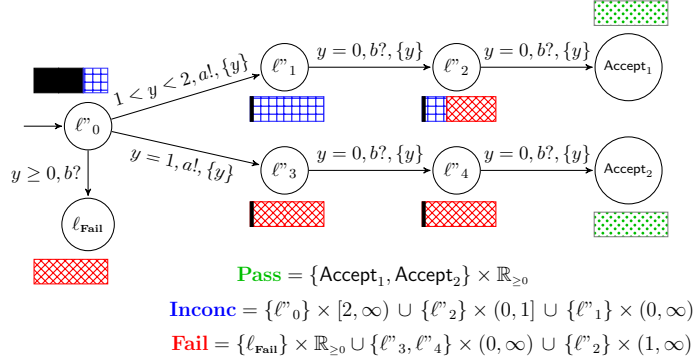
Selection of $\mathcal{T}C$: So far, the construction of $\mathcal{T}C'$ determines **Verdicts**, but does not perform any selection of behaviors. A last step consists in trying to control the behavior of $\mathcal{T}C'$ in order to avoid **Inconc** states (thus stay in $\text{pref}(\text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P}))$), because reaching **Inconc**


 Figure 16: Test case \mathcal{T}' with verdicts

means that **Pass** is unreachable, thus \mathcal{TP} cannot be satisfied anymore. To this aim, guards of transitions of \mathcal{T}' are refined in the final test case \mathcal{T} in two complementary ways. First, transitions leaving a verdict state (**Fail**, **Inconc** or **Pass**) are useless, because the test case execution stops when a verdict is issued. Thus for each transition, the guard is intersected with the predicate characterizing the set of valuations associated with **None** in the source location. This does not change the verdict of traces. Second, transitions arriving in **Inconc** states and carrying outputs can be avoided (outputs are controlled by the test case), thus for any transition labeled by an output, the guard is intersected with the predicate characterizing **None** and **Pass** states in the target location (*i.e.* states that are not in **Inconc**, as **Fail** cannot be reached by an output). The effect is to suppress some traces leading to **Inconc** states. All in all, traces in \mathcal{T} are exactly those of \mathcal{T}' that traverse only **None** states (except for the last state), and do not end in **Inconc** with an output. This selection does not impact on the properties of test suites (soundness, strictness, precision and exhaustiveness) as will be seen later.

Example 4.7. Figure 17 represents the test case obtained after this selection phase. One can notice that locations ℓ''_{11} , ℓ''_{12} , ℓ''_{13} and ℓ''_{21} , ℓ''_{22} have been removed since they can only be reached from **Inconc** states, thus a verdict will have been emitted before reaching those locations. The avoidance of **Inconc** verdicts by outputs cannot be observed on this example. However, with a small modification of \mathcal{A} consisting in adding initially the reception of an a before one time unit, and not followed by two b 's but *e.g.* one c , the resulting transition labeled with $(0 \leq y < 1, a!)$ in \mathcal{T}' could be cut, producing the same \mathcal{T} .

Remark 4.8. Notice that in the example, falling into **Inconc** in ℓ''_0 could be avoided by adding the invariant $y < 2$, with the effect of forcing to output a . More generally, invariants can be added to locations by rendering outputs urgent in order to avoid **Inconc**, while taking care of keeping test cases non-blocking, *i.e.* by ensuring that an output can be done just before the invariant becomes false. More precisely, $I(\ell)$ is the projection of **None** on ℓ if **Inconc** is reachable by letting time elapse and it preserves the non-blocking property, **true** otherwise.

Figure 17: Final test case \mathcal{TC} after selection

Complexity. Let us discuss the complexity of the construction of \mathcal{TC} from \mathcal{DP} . Note that the size of TAIIO \mathcal{TC} is linear in the size of \mathcal{DP} but the difficulty lies in the computation of **Verdicts**. Computing **Pass** is immediate. The set $\mathbf{coreach}(\mathbf{Pass})$ can be computed in polynomial time (more precisely in $\mathcal{O}(|L^{\mathcal{DP}}| \cdot |X^{\mathcal{DP}}| \cdot |M^{\mathcal{DP}}|)$). To explain this, observe that guards in the TAIIO \mathcal{DP} are regions and with each location ℓ is associated an initial region r_ℓ such that guards of transitions leaving ℓ are time successors of r_ℓ . Thus during the computation of $\mathbf{coreach}(\mathbf{Pass})$, for each location ℓ , one only needs to consider these $\mathcal{O}(|X^{\mathcal{DP}}| \cdot |M^{\mathcal{DP}}|)$ different regions in order to determine the latest time-successor r_ℓ^{\max} of r_ℓ which is co-reachable from **Pass**. Then **None** states with location ℓ are exactly those within regions that are time-predecessors of r_ℓ^{\max} . For the same reason (number of possible guards outgoing a given location) $E_{\ell_{\text{Fail}}}$ can be computed in polynomial time. Last the **Fail** verdicts in locations (except for ℓ_{Fail}) are computed in linear time by complementing the invariants in \mathcal{DP} . The test selection can be done by inspecting all transitions: a transition is removed if either the source state is a verdict state, or it corresponds to an output action and the successor are **Inconc** states. This last step thus only requires linear time. To conclude, the overall complexity of construction of \mathcal{TC} from \mathcal{DP} is polynomial.

4.3. Test suite properties. We have presented the different steps for the generation of a TAIIO test case from a TAIIO specification and an OTAIO test purpose. The following results express their properties.

Theorem 4.9. *Any test case \mathcal{TC} built by the procedure is sound for \mathcal{A} . Moreover, if \mathcal{DP} is an exact approximation of \mathcal{P} (i.e. $\text{Traces}(\mathcal{DP}) = \text{Traces}(\mathcal{P})$), the test case \mathcal{TC} is also strict and precise for \mathcal{A} and \mathcal{TP} .*

The proof is detailed below, but we first give some intuition. As a preamble, notice that, as explained in the paragraph on test selection, traces of \mathcal{TC}' are not affected by the construction of \mathcal{TC} . In particular, the transitions considered in the proof are identical in \mathcal{TC} and \mathcal{TC}' . Soundness comes from the construction of $E_{\ell_{\text{Fail}}}$ in \mathcal{TC} and preservation of soundness by the approximate determinization \mathcal{DP} of \mathcal{P} given by Corollary 2.10. When \mathcal{DP} is an exact determinization of \mathcal{P} , \mathcal{DP} and \mathcal{P} have same traces, which also equal traces of \mathcal{A} since \mathcal{TP} is complete. Strictness then comes from the fact that \mathcal{DP} and \mathcal{A} have the same non-conformant traces, which are captured by the definition of $E_{\ell_{\text{Fail}}}$ in \mathcal{TC} . Precision comes from $\text{Traces}_{\text{Accept}^{\mathcal{DP}}}(\mathcal{DP}) = \text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P})$ and from the definition of **Pass**.

When \mathcal{DP} is not exact however, there is a risk that some behaviors allowed in \mathcal{DP} are not in \mathcal{P} , thus some non-conformant behaviors are not detected, even if they are executed by \mathcal{TC} . Similarly, some **Pass** verdicts may be produced for non-accepted or even non-conformant behaviors. However, if a trace in $\text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P})$ is present in \mathcal{TC} and observed during testing, a **Pass** verdict will be delivered. In other words, precision is not always satisfied, but the “only if” direction of precision (Definition 4.3) is satisfied.

Proof. Soundness: To prove soundness, we need to show that for any $\mathcal{I} \in \mathcal{I}(\mathcal{A})$, \mathcal{I} fails \mathcal{TC} implies $\neg(\mathcal{I} \text{ tioco } \mathcal{A})$.

Assuming that \mathcal{I} fails \mathcal{TC} , there exists a trace $\sigma \in \text{Traces}(\mathcal{I}) \cap \text{Traces}_{\text{Fail}}(\mathcal{TC})$. By the construction of the set **Fail** in \mathcal{TC} , there are two cases: either σ leads to a location $(\ell, \neg(I(\ell)))$ in \mathcal{DP} , or σ leads to a state with location ℓ_{Fail} . In the first case, $\sigma = \sigma' \cdot \delta$ where $\sigma' \in \text{Traces}(\mathcal{DP})$ and $\delta > 0$ violates the invariant in the location of \mathcal{DP} after σ' , and in the second case, by the construction of $E_{\ell_{\text{Fail}}}$, $\sigma = \sigma' \cdot a$ where $\sigma' \in \text{Traces}(\mathcal{DP})$ and $a \in \Sigma_{\uparrow}^{\mathcal{P}}$ is unspecified in \mathcal{DP} after σ' . In both cases, by definition, this means that $\neg(\mathcal{I} \text{ tioco } \mathcal{DP})$, which proves that \mathcal{TC} is sound for \mathcal{DP} . Now, as \mathcal{DP} is an io-abstraction of \mathcal{P} (i.e. $\mathcal{P} \preceq \mathcal{DP}$), by Corollary 2.10 this entails that \mathcal{TC} is sound for \mathcal{P} . Finally, we have $\text{Traces}(\mathcal{P}) = \text{Traces}(\mathcal{A})$, which trivially implies that $\mathcal{A} \preceq \mathcal{P}$, and thus that \mathcal{TC} is also sound for \mathcal{A} .

Strictness: For strictness, in the case where \mathcal{DP} is an exact approximation of \mathcal{P} , we need to prove that for any $\mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\neg(\mathcal{I} \parallel \mathcal{TC} \text{ tioco } \mathcal{A})$ implies that \mathcal{I} fails \mathcal{TC} . Suppose that $\neg(\mathcal{I} \parallel \mathcal{TC} \text{ tioco } \mathcal{A})$. By definition, there exists $\sigma \in \text{Traces}(\mathcal{A})$ and $a \in \text{out}(\mathcal{I} \parallel \mathcal{TC} \text{ after } \sigma)$ such that $a \notin \text{out}(\mathcal{A} \text{ after } \sigma)$. Since \mathcal{DP} is an exact approximation of \mathcal{P} , we have the equalities $\text{Traces}(\mathcal{DP}) = \text{Traces}(\mathcal{P}) = \text{Traces}(\mathcal{A})$, thus $\sigma \in \text{Traces}(\mathcal{DP})$ and $a \notin \text{out}(\mathcal{DP} \text{ after } \sigma)$. By construction of **Fail** in \mathcal{TC} , it follows that $\sigma \cdot a \in \text{Traces}_{\text{Fail}}(\mathcal{TC})$ which, together with $\sigma \cdot a \in \text{Traces}(\mathcal{I})$, implies that \mathcal{I} fails \mathcal{TC} . Thus \mathcal{TC} is strict.

Precision: To prove precision, in the case of exact determinization, we have to show that for any trace σ , $\text{Verdict}(\sigma, \mathcal{TC}) = \text{Pass} \iff \sigma \in \text{Traces}(\text{Seq}_{\text{Accept}^{\mathcal{TP}}}(\mathcal{TP}) \cap \text{Seq}(\mathcal{A}))$. The definition of **Pass** = $\bigcup_{\ell \in \text{Accept}^{\mathcal{DP}}}(\{\ell\} \times I^{\mathcal{DP}}(\ell))$ in \mathcal{TC} implies that a **Pass** verdict is produced for σ exactly when $\sigma \in \text{Traces}_{\text{Accept}^{\mathcal{DP}}}(\mathcal{DP})$ which equals $\text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P}) = \text{Traces}(\text{Seq}_{\text{Accept}^{\mathcal{TP}}}(\mathcal{TP}) \cap \text{Seq}(\mathcal{A}))$ when \mathcal{DP} is exact. \square

Example 4.10. The test case \mathcal{TC} of Figure 17 comes from an approximate determinization. However, the approximation comes after reaching **Inconc** states. More precisely, in the gray state of the game in Figure 14, the approximation starts in the time interval $(2, \infty)$. This state corresponds to location ℓ''_1 in \mathcal{TC} where the verdict is **Inconc** as soon as a non null delay is observed. The test case is thus strict and precise, despite the over-approximation in the determinization phase.

In the following, we prove an exhaustiveness property of our test generation method when determinization is exact. For technical reasons, we need to restrict to a sub-class of TAIOS defined below. We discuss this restriction later.

Definition 4.11. We say that an OTAIO \mathcal{A} is *repeatedly observable* if from any state of \mathcal{A} , there is a future observable transition, i.e. $\forall s \in S^{\mathcal{A}}$, there exists μ such that $s \xrightarrow{\mu}$ and $\text{Trace}(\mu) \notin \mathbb{R}_{\geq 0}$.

Theorem 4.12 (Exhaustiveness). *Let \mathcal{A} be a repeatedly observable TAIIO which can be exactly determinized by our approach. Then the set of test cases that can be generated from \mathcal{A} by our method is exhaustive.*

Proof. Let $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_{\gamma}^{\mathcal{A}}, \Sigma_{\dagger}^{\mathcal{A}}, \Sigma_{\tau}^{\mathcal{A}}, X_p^{\mathcal{A}}, \emptyset, M^{\mathcal{A}}, I^{\mathcal{A}}, E^{\mathcal{A}})$ be the TAIIO specification, and $\mathcal{I} = (L^{\mathcal{I}}, \ell_0^{\mathcal{I}}, \Sigma_{\gamma}^{\mathcal{I}}, \Sigma_{\dagger}^{\mathcal{I}}, \Sigma_{\tau}^{\mathcal{I}}, X_p^{\mathcal{I}}, \emptyset, M^{\mathcal{I}}, I^{\mathcal{I}}, E^{\mathcal{I}})$ any non-conformant implementation in $\mathcal{I}(\mathcal{A})$. The idea is now to prove that from \mathcal{A} and \mathcal{I} , one can build a test purpose \mathcal{TP} such that the test case \mathcal{TC} built from \mathcal{A} and \mathcal{TP} may detect this non-conformance, *i.e.* \mathcal{I} fails \mathcal{TC} .

By definition of $\neg(\mathcal{I} \text{ tioco } \mathcal{A})$, there exists $\sigma \in \text{Traces}(\mathcal{A})$ and $a \in \Sigma_{\dagger}^{\mathcal{A}} \sqcup \mathbb{R}_{\geq 0}$ such that $a \in \text{out}(\mathcal{I} \text{ after } \sigma)$ but $a \notin \text{out}(\mathcal{A} \text{ after } \sigma)$. Since \mathcal{A} is repeatedly observable, there also exists $\delta \in \mathbb{R}_{\geq 0}$ and $b \in \Sigma_{obs}^{\mathcal{A}}$ such that $\sigma.\delta.b \in \text{Traces}(\mathcal{A})$.

As \mathcal{A} can be determinized exactly by our approach, there must exist some resources (k, M) and a strategy Π for Determinizator in the game $\mathcal{G}_{\mathcal{A},(k,M)}$ such that $\text{Traces}(\text{Aut}(\Pi)) = \text{Traces}(\mathcal{A})$.

From the non-conformant implementation \mathcal{I} , a test purpose $(\mathcal{TP}, \text{Accept}^{\mathcal{TP}})$ can be built, with $\mathcal{TP} = (L^{\mathcal{TP}}, \ell_0^{\mathcal{TP}}, \Sigma_{\gamma}^{\mathcal{A}}, \Sigma_{\dagger}^{\mathcal{A}}, \Sigma_{\tau}^{\mathcal{A}}, X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}}, M^{\mathcal{TP}}, I^{\mathcal{TP}}, E^{\mathcal{TP}})$, $X_p^{\mathcal{TP}} = X_p^{\mathcal{I}} \sqcup X^{\text{Aut}(\Pi)}$ and $X_o^{\mathcal{TP}} = \emptyset$, and $\sigma.\delta.b \in \text{Traces}_{\text{Accept}^{\mathcal{TP}}}(\mathcal{TP})$ but none of its prefixes is in $\text{Traces}_{\text{Accept}^{\mathcal{TP}}}(\mathcal{TP})$. The construction of \mathcal{TP} relies on the region graph of $\mathcal{I} \parallel \text{Aut}(\Pi)$. First a TAIIO \mathcal{TP}' is built which recognizes exactly the traces read along the path corresponding to σ in the region graph of $\mathcal{I} \parallel \text{Aut}(\Pi)$, followed by a transition b with the guard corresponding to the one in $\text{Aut}(\Pi)$. In particular it recognizes the trace $\sigma.\delta.b$. The test purpose $(\mathcal{TP}, \text{Accept}^{\mathcal{TP}})$ is then built such that \mathcal{TP} accepts in its states $\text{Accept}^{\mathcal{TP}}$ the traces of \mathcal{TP}' . Note that \mathcal{TP} should be complete for Σ , thus locations of \mathcal{TP}' should be completed by adding loops without resets for all actions in Σ_{τ} , and adding, for all observable actions, transitions to a trap location guarded with negations of their guards in \mathcal{TP}' .

Now consider our test generation method applied to \mathcal{TP} and \mathcal{A} . First $\mathcal{P} = \mathcal{A} \times \mathcal{TP}$ is built, and we consider the game $\mathcal{G}_{\mathcal{A},(k',M')}$ with $k' = k + |X_p^{\mathcal{TP}}|$ and $M' = \max(M, M^{\mathcal{TP}})$. One can then define a strategy Π' composed of the strategy Π for the k first clocks, and following the resets of \mathcal{TP} (which is deterministic) for the other clocks corresponding to those in $X_p^{\mathcal{TP}}$. The construction of $(\mathcal{DP}, \text{Accept}^{\mathcal{DP}})$ following the strategy Π' thus ensures that $\text{Traces}(\mathcal{DP}) = \text{Traces}(\mathcal{P})$ and $\text{Traces}_{\text{Accept}^{\mathcal{DP}}}(\mathcal{DP}) = \text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P})$.

Finally, let \mathcal{TC} be the test case built from \mathcal{DP} . Observe that \mathcal{TC} after $\sigma.\delta.b \subseteq \mathbf{Pass}$, but \mathcal{TC} after $\sigma.\delta \not\subseteq \mathbf{Pass}$. As a consequence, \mathcal{TC} after $\sigma \subseteq \mathbf{None}$. Moreover we have $a \notin \text{out}(\mathcal{A} \text{ after } \sigma)$, hence $\sigma.a \in \text{Traces}_{\mathbf{Fail}}(\mathcal{TC})$ and as $\sigma.a \in \text{Traces}(\mathcal{I})$, we can conclude that \mathcal{I} fails \mathcal{TC} . \square

Discussion: The hypothesis that \mathcal{A} is repeatedly observable is in fact not restrictive for a TAIIO that is determinizable by our approach. Indeed, such a TAIIO can be transformed into a repeatedly observable one with same conformant implementations, by first determinizing it, and then completing it as follows. In all locations, a transition labeled by an input is added, which goes to a trap state looping for all outputs, and is guarded by the negation of the union of guards of transitions for this input in the deterministic automaton.

When \mathcal{A} cannot be determinized exactly, the risk is that some non-conformance may be undetectable. However, the theorem can be generalized to non-determinizable automata with no resets on internal action. Indeed, in this case, in the game with resources (k, M) , where k is the length of the finite non-conformant trace $\sigma.a$, the strategy consisting in resetting a new clock at each observable action allows to remain exact until the observation of non-conformance (see remark after Theorem 3.2). The proof of theorem 4.12 can be adapted using this strategy.

5. DISCUSSION AND RELATED WORK

Alternative definitions of test purposes. The definition of test purposes depends on the semantic level at which behaviors to be tested are described (*e.g.* sequences, traces). This induces a trade-off between the precision of the description of behaviors, and the cost of producing test suites. In this work, test purposes recognize timed sequences of the specification \mathcal{A} , by a synchronization with actions and observed clocks. They also have their own proper clocks for additional precision. The advantage is a fine tuning of selection. The price to be paid is that, for each test purpose, the whole sequence of operations, including determinization which may be costly, must be done. An alternative is to define test purposes recognizing timed traces rather than timed sequences. In this case, selection should be performed on a deterministic io-abstraction \mathcal{B} of \mathcal{A} obtained by an approximate determinization of \mathcal{A} . Then, test purposes should not refer to \mathcal{A} 's clocks as these are lost by the approximate determinization. Test purposes should then either observe \mathcal{B} 's clocks, and thus be defined after determinization, or use only proper clocks in order not to depend on \mathcal{B} , at the price of further restricting the expressive power of test purposes. In both cases, test purposes should preferably be deterministic in order to avoid a supplementary determinization after the product with \mathcal{B} . The main advantage of these approaches is that the specification is determinized only once, which reduces the cost of producing a test suite. However, the expressive power of test purposes is reduced.

Test execution. Once test cases are selected, it remains to execute them on a real implementation. As a test case is a TAIIO, and not a simple timed trace, a number of decisions still need to be taken at each state of the test case: (1) whether to wait for a certain delay, or to receive an input or to send an output (2) which output to send, in case there is a choice. It is clear that different choices may lead to different behaviors and verdicts. Some of these choices can be made either randomly (*e.g.* choosing a random time delay, choosing between outputs, etc), or can be pre-established according to user-defined strategies. One such policy is to apply a technique similar to the control approach of [10] whose goal is to avoid $\text{RTraces}(\mathcal{A}, \mathcal{TP})$.

Moreover, the tester's time observation capabilities are limited in practice: testers only dispose of a finite-precision digital clock (a counter) and cannot distinguish among observations which elude their clock precision. Our framework may take this limitation into account. In [18] assumptions on the tester's digital clock are explicitly modeled as a special TAIIO called *Tick*, synchronized with the specification before test generation, then relying to the untimed case. We could imagine to use such a *Tick* automaton differently, by synchronizing it with the resulting test case after generation.

Related work. As mentioned in the introduction, off-line test selection is in general restricted to deterministic automata or known classes of determinizable timed automata. An exception is the work of [18] which relies on an over-approximate determinization. Compared to this work, our approximate determinization is more precise (it is exact in more cases), it copes with outputs and inputs using over- and under-approximations, and preserves urgency in test cases as much as possible. Another exception is the work of [10], where the authors propose a game approach whose effect can be understood as a way to completely avoid $\text{RTraces}(\mathcal{A}, \mathcal{TP})$, with the possible risk of missing some or even all traces

in $\text{pref}(\text{Traces}_{\text{Accept}^{\mathcal{P}}}(\mathcal{P}))$. Our selection, which allows to lose this game and produce an **Inconc** verdict when this happens, is both more liberal and closer to usual practice.

In several related works [16, 11], test purposes are used for test case selection from TAIOS. In all these works, test purposes only have proper clocks, thus cannot observe clocks of the specification.

It should be noticed that selection by test purposes can be used for test selection with respect to coverage criteria [26]. Those coverage criteria define a set of elements (generally syntactic ones) to be covered (*e.g.* locations, transitions, branches, etc). Each element can then be translated into a test purpose, the produced test suite covering the given criteria.

6. CONCLUSION

In this article, we presented a complete formalization and operations for the automatic off-line generation of test cases from non-deterministic timed automata with inputs and outputs (TAIOS). The model of TAIOS is general enough to take into account non-determinism, partial observation and urgency. One main contribution is the ability to tackle any TAIOS, thanks to an original approximate determinization algorithm. Another main contribution is the selection of test cases with expressive test purposes described as OTAIOS having the ability to precisely select behaviors to be tested based on clocks and actions of the specification as well as proper clocks. Test cases are generated as TAIOS using a symbolic co-reachability analysis of the observable behaviors of the specification guided by the test purpose.

A first perspective of this work is to implement the approach in a test generation tool. Currently, the approximate determinization has been prototyped in Python thanks to a binding of the UPPAAL DBM library [25]. Other perspectives could be to combine this approach with the one of [14] for models with data, for the generation of test cases from models with both time and data in the spirit of [3], but generalized to non-deterministic models.

Acknowledgements: we would like to thank the reviewers for their constructive comments that allowed us to improve this article.

REFERENCES

- [1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [2] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *LNCS*, pages 163–178, 1998.
- [3] W. L. Andrade, P. Machado, T. Jéron, and H. Marchand. Abstracting time and data for conformance testing of real-time systems. In *7th Workshop on Advances in Model Based Testing (A-MOST'11)*, Berlin, Germany, March 2011.
- [4] B. Bérard, P. Gastin, and A. Petit. On the power of non-observable actions in timed automata. In *13th Annual Symposium on Theoretical Aspects of Computer Science (STACS'96)*, volume 1046 of *LNCS*, pages 255–268, 1996.
- [5] N. Bertrand, T. Jéron, A. Stainer, and M. Krichen. Off-line test selection with test purposes for non-deterministic timed automata. In *17th International Conference on Tools and Algorithms for the Construction And Analysis of Systems (TACAS'11)*, volume 6605 of *LNCS*, pages 96–111. Springer, 2011. Extended version as INRIA report 7501, <http://hal.inria.fr/inria-00550923>.

- [6] N. Bertrand, A. Stainer, T. Jéron, and M. Krichen. A game approach to determinize timed automata. Technical Report 7381, INRIA, september 2010, <http://hal.inria.fr/inria-00524830>.
- [7] N. Bertrand, A. Stainer, T. Jéron, and M. Krichen. A game approach to determinize timed automata. In *14th International Conference on Foundations of Software Science and Computation Structures (FOSACS'11)*, volume 6604 of *LNCS*, pages 245–259. Springer, 2011.
- [8] L. B. Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In *4th International Workshop on Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *LNCS*, pages 64–78. Springer, 2005.
- [9] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed I/O automata: a complete specification theory for real-time systems. In *13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC'10)*, pages 91–100. ACM Press, 2010.
- [10] A. David, K. G. Larsen, S. Li, and B. Nielsen. Timed testing under partial observability. In *2nd International Conference on Software Testing Verification and Validation (ICST'09)*, pages 61–70. IEEE Computer Society, 2009.
- [11] A. En-Nouaary and R. Dssouli. A guided method for testing timed input output automata. In *15th IFIP International Conference on Testing of Communicating Systems (TestCom'03)*, volume 2644 of *LNCS*, pages 211–225, 2003.
- [12] O. Finkel. Undecidable problems about timed automata. In *4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *LNCS*, pages 187–199, 2006.
- [13] C. Jard and T. Jéron. TGV: theory, principles and algorithms. *Software Tools for Technology Transfer*, 7(4):297–315, 2005.
- [14] B. Jeannet, T. Jéron, V. Rusu, and E. Zinovieva. Symbolic test selection based on approximate analysis. In *11th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, volume 3440 of *LNCS*, pages 349–364. Springer, April 2005.
- [15] A. Khoumsi, T. Jéron, and H. Marchand. Test cases generation for nondeterministic real-time systems. In *Formal Approaches to Software Testing (FATES'03)*, volume 2931 of *LNCS*, pages 131–145, 2004.
- [16] O. Koné, R. Castanet, and P. Laurencot. On the fly test generation for real time protocols. In *7th International Conference on Computer Communications & Networks (IC3N'98)*, pages 378–387. IEEE, 1998.
- [17] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *11th International SPIN Workshop (SPIN'04)*, volume 2989 of *LNCS*, pages 109–126. Springer, 2004.
- [18] M. Krichen and S. Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, 2009.
- [19] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using Uppaal. In *4th International Workshop on Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *LNCS*, pages 79–94. Springer, 2005.
- [20] R. Mazala. Infinite games. In *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*, pages 23–42. Springer, 2002.
- [21] B. Nielsen and A. Skou. Automated test generation from timed automata. *Software Tools for Technology Transfer*, 5(1):59–77, 2003.
- [22] J. Schmaltz and J. Tretmans. On conformance testing for timed systems. In *6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *LNCS*, pages 250–264. Springer, 2008.
- [23] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.
- [24] S. Tripakis. Folk theorems on the determinization and minimization of timed automata. *Information Processing Letters*, 99(6):222–226, 2006.
- [25] UPPAAL DBM Library, <http://people.cs.aau.dk/~adavid/UDBM/python.html>.
- [26] H. Zhu, P. A. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, 1997.