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

Nathalie Bertrand[1], Thierry Jéron[1], Amélie Stainer[1], Moez Krichen[2]

[1] INRIA Rennes - Bretagne Atlantique, Rennes, France
[2] Institute of Computer Science and Multimedia, Sfax, Tunisia

Abstract. This paper proposes novel off-line test generation techniques for 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, but is in general impossible. This problem is solved here thanks to an approximate determinization using a game approach, which preserves **tioco** and guarantees the soundness of generated test cases. A second problem is test selection for which a precise description of timed behaviors to be tested is carried out by expressive test purposes modeled by a generalization of TAIOs. Finally, using a symbolic co-reachability analysis guided by the test purpose, test cases are generated in the form of TAIOs equipped with verdicts.

Keywords: Conformance testing, timed automata, partial observability, urgency, approximate determinization, game, test purpose

1 Introduction

Conformance testing is the process of testing whether an implementation behaves correctly with respect to a specification. 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 *model-based conformance testing* models are used to describe testing artifacts (specifications, implementations, test cases, ...), conformance is formally defined and test cases with verdicts are generated automatically, thus improving the quality of testing by ensuring properties relating verdicts of executions of test cases with conformance. For timed models, model-based conformance testing has already been explored in the last decade, with different models and conformance relations (see *e.g.* [14] for a survey), and test generation algorithms (e.g. [5, 12, 13]). 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 partially observable and non-deterministic TAIOs with invariants for the modeling of urgency.

One of the main difficulties encountered in test generation for those partially observable, non-deterministic TAIOs is determinization which is required in order to foresee the next enabled actions during execution, and thus to emit a

* This work was partly funded by the french ANR project Testec.

correct verdict. In fact TAs (and thus TAIOs) are not determinizable in general [1]. This motivated two different approaches for test generation from timed models in the literature. In *off-line test generation* test cases are first generated as timed automata (or timed sequences, or trees) 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. [10, 13]), except in [12] where the problem is solved by the use of an over-approximate determinization with fixed resources. In *on-line test generation*, test cases are generated during their execution. This can be applied to any TAIO as determinization is then performed along the current finite execution. It is of particular interest to rapidly discover errors, 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 delays).

In this paper, we propose to generate test cases off-line for general 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 [12], using a game approach [4]. Determinization is exact for all 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, approximate determinization guarantees soundness of generated test cases by producing a deterministic *io-abstraction* of the TAIO for a particular *io-refinement* relation, generalizing the io-refinement for deterministic TAIOs of [6]. Our method is more precise than [12] (see [4] for details) and preserves the richness of our model by dealing with partial observability and urgency. Behaviors of specifications to be tested are identified by means of test purposes. These are defined as *open timed automata with inputs and outputs* (OTAIOs), a model generalizing TAIOs, allowing to precisely describe behaviors according to actions and clocks of the specification as well as proper clocks. Then test selection is performed by a co-reachability analysis, and produces a test case in the form of a TAIO. To our knowledge, this work constitutes the most general and advanced off-line test selection approach for TAIOs.

The paper is structured as follows. In the next section we introduce the model of OTAIOs, its semantics, some notations and operations. Section 3 recalls the **tioco** conformance theory including expected properties relating conformance and verdicts, and an io-refinement relation preserving **tioco**. Section 4 presents our game approach for the approximate determinization compatible with the io-refinement. In Section 5 we detail the test selection mechanism using test purposes and prove some properties on generated test cases.

2 A model of open timed automata with inputs/outputs

In this section, we first introduce the model of *open timed automata with inputs/outputs* (OTAIOs for short), a very general model for open, non-deterministic and time constrained systems that will serve to model all testing artifacts. This

model is an extension of timed automata with inputs and outputs (TAIOs), itself a usual extension of timed automata (TAs) [1]. The extension consists in a partition of the set of actions into inputs, outputs and unobservable actions, and a partition of the set of clocks into proper and observed clocks that will be useful for the definition of test purposes. Like in [1] OTAIOs have location invariants for the modeling of urgency.

2.1 Timed automata with inputs/outputs

We start by introducing notations and useful definitions concerning TAs.

Given X a finite set of *clocks*, a *clock valuation* is a mapping $v : X \to \mathbb{R}_+$. If v is a valuation over X and $t \in \mathbb{R}$, 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 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 $x \lhd c$ where $x \in X, c \in [0, M] \cap \mathbb{N}$ and $\lhd \in \{<, \leq\}$. We denote by $G_M(X)$ (resp. $I_M(X)$) the set of M-bounded guards (resp. invariants) over X.

Definition 1 (OTAIO). *An* open timed automaton with inputs and outputs *(OTAIO) is a tuple* $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{A}}, X_o^{\mathcal{A}}, M^{\mathcal{A}}, I^{\mathcal{A}}, E^{\mathcal{A}})$ *such that:*

- $L^{\mathcal{A}}$ *is a finite set of* locations*, with* $\ell_0^{\mathcal{A}} \in L^{\mathcal{A}}$ *the* initial location*,*
- $\Sigma_?^{\mathcal{A}}$*,* $\Sigma_!^{\mathcal{A}}$ *and* $\Sigma_\tau^{\mathcal{A}}$ *are disjoint finite* alphabets *of* input actions *(noted* $a?, b?, \ldots$*),* output actions *(noted* $a?, b?, \ldots$*), and* internal actions *(noted* $\tau_1, \tau_2, \ldots$*). We note* $\Sigma_{obs}^{\mathcal{A}} = \Sigma_?^{\mathcal{A}} \sqcup \Sigma_!^{\mathcal{A}}$ *for the alphabet of observable actions, and* $\Sigma^{\mathcal{A}} = \Sigma_?^{\mathcal{A}} \sqcup \Sigma_!^{\mathcal{A}} \sqcup \Sigma_\tau^{\mathcal{A}}$ *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* $\mathcal{A}$*,*
- $I^{\mathcal{A}} : L^{\mathcal{A}} \to I_{M^{\mathcal{A}}}(X^{\mathcal{A}})$ *is a mapping labeling each location with an* invariant*,*
- $E^{\mathcal{A}} \subseteq L^{\mathcal{A}} \times G_{M^{\mathcal{A}}}(X^{\mathcal{A}}) \times \Sigma^{\mathcal{A}} \times 2^{X_p^{\mathcal{A}}} \times L^{\mathcal{A}}$ *is a finite set of* edges *where* guards *are defined on* $X^{\mathcal{A}}$*, but* resets *are restricted to proper clocks in* $X_p^{\mathcal{A}}$*.*

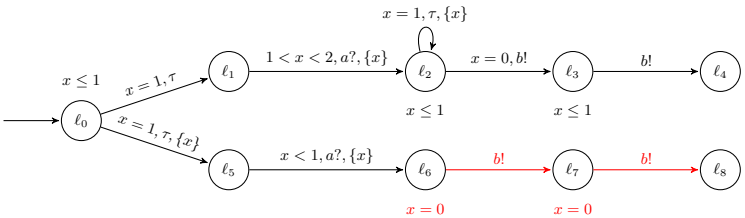

Fig. 1. Specification $\mathcal{A}$

The reason for introducing the OTAIO model is to have a unique model (syntax and semantics) that will be next specialized for particular testing artifacts. In particular, an OTAIO with an empty set of observed clocks $X_o^{\mathcal{A}}$ is a classical TAIO, and will be the model for specifications, implementations and test cases. For example, Fig. 1 represents such a TAIO for a specification $\mathcal{A}$ with clock x, input a, output b and internal action τ. The partition of actions reflects their roles in the testing context: the environment 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 $\mathcal{A}$, and *observed clocks* referring to proper clocks of another OTAIO. These cannot be reset to avoid intrusiveness, but synchronization with them in guards and invariants is allowed. In particular, test purposes have observed clocks which observe proper clocks of specifications in order to describe time constrained behaviors to be tested.

2.2 The semantics of OTAIOs

The semantics of an OTAIO $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?^{\mathcal{A}}, \Sigma_!^{\mathcal{A}}, \Sigma_\tau^{\mathcal{A}}, X_p^{\mathcal{A}}, X_o^{\mathcal{A}}, M^{\mathcal{A}}, I^{\mathcal{A}}, E^{\mathcal{A}})$ is a timed transition system $\mathcal{T}^{\mathcal{A}} = (S^{\mathcal{A}}, s_0^{\mathcal{A}}, \Gamma^{\mathcal{A}}, \rightarrow_{\mathcal{A}})$ where $S^{\mathcal{A}} = L^{\mathcal{A}} \times \mathbb{R}_+^{X^{\mathcal{A}}}$ is the set of *states i.e.* pairs (ℓ, v) consisting in a location and a valuation of clocks; $s_0^{\mathcal{A}} = (\ell_0^{\mathcal{A}}, \overline{0}) \in S^{\mathcal{A}}$ is the *initial state*; $\Gamma^{\mathcal{A}} = \mathbb{R}_+ \cup E^{\mathcal{A}} \times 2^{X_o^{\mathcal{A}}}$ is the set of transition *labels* consisting in either a delay δ or a pair (e, X'_o) formed by an edge and a set of observed clocks; the transition relation $\rightarrow_{\mathcal{A}} \subseteq S^{\mathcal{A}} \times \Gamma^{\mathcal{A}} \times S^{\mathcal{A}}$ is the smallest set of the following moves:

- *Discrete moves:* $(\ell, v) \xrightarrow{(e, X'_o)}_{\mathcal{A}} (\ell', v')$ whenever there exists $e = (\ell, g, a, X'_p, \ell') \in E^{\mathcal{A}}$ such that $v \models g \wedge I^{\mathcal{A}}(\ell)$, $X'_o \subseteq X_o^{\mathcal{A}}$ is an arbitrary subset of observed clocks, $v' = v_{[X'_p \cup X'_o \leftarrow 0]}$ and $v' \models I^{\mathcal{A}}(\ell')$. Note that X'_o is unconstrained as observed clocks are controlled by another OTAIO.
- *Time elapse:* $(\ell, v) \xrightarrow{\delta}_{\mathcal{A}} (\ell', v + \delta)$ for $\delta \in \mathbb{R}_+$ if $v + \delta \models I^{\mathcal{A}}(\ell)$.

A *partial run* of $\mathcal{A}$ is a finite sequence of subsequent moves in $(S^{\mathcal{A}} \times \Gamma^{\mathcal{A}})^*.S^{\mathcal{A}}$. 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 $s_0^{\mathcal{A}}$. $\mathsf{Run}(\mathcal{A})$ and $\mathsf{pRun}(\mathcal{A})$ will denote respectively runs and partial runs of $\mathcal{A}$. A state s is *reachable* if there exists a run leading to s. A state s is *co-reachable* from a set $S' \subseteq S^{\mathcal{A}}$ if there is a partial run from s to a state in S'. We note $\mathtt{reach}(\mathcal{A})$ the set of reachable states and $\mathtt{coreach}(\mathcal{A}, S')$ 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. 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 $\mu = \delta_1.(a_1, X_p^1 \cup X_o^1) \cdots \delta_k.(a_k, X_p^k \cup X_o^k)$ where $\forall i \in [1, k], e_i = (\ell_i, g_i, a_i, X_p^i, \ell'_i)$. We then note $s_0^{\mathcal{A}} \xrightarrow{\mu}_{\mathcal{A}} s_k$. We write $s_0^{\mathcal{A}} \xrightarrow{\mu}_{\mathcal{A}}$ for $\exists s_k, s_0^{\mathcal{A}} \xrightarrow{\mu}_{\mathcal{A}} s_k$. We note $\mathsf{Seq}(\mathcal{A}) \subseteq (\mathbb{R}_+ \cup (\Sigma^{\mathcal{A}} \times 2^{X^{\mathcal{A}}}))^*$ (respectively $\mathsf{pSeq}(\mathcal{A})$) the set of sequences (resp. partial sequences) of $\mathcal{A}$. For a sequence μ, we note $time(\mu)$ the sum of delays in μ.

For a (partial) sequence $\mu \in \mathsf{pSeq}(\mathcal{A})$, $Trace(\mu)$ denotes the observable timed word in $(\mathbb{R}_+ \cup \Sigma^{\mathcal{A}}_{obs})^*.\mathbb{R}_+$ obtained by erasing from μ all internal actions, and summing delays between observable actions (if any). It is defined inductively as follows: $Trace(\varepsilon) = 0$, $Trace((\tau, X).\mu) = Trace(\mu)$, $Trace(\delta_1 \dots \delta_k) = \Sigma_{i=1}^k \delta_i$, and $Trace(\delta_1 \dots \delta_k.(a, X').\mu) = (\Sigma_{i=1}^k \delta_i).a.Trace(\mu)$ if $a \in \Sigma^{\mathcal{A}}_{obs}$. For example $Trace(1.(\tau, X^1).2.(a, X^2).2.(\tau, X^3)) = (3, a).2$ and $Trace(1.(\tau, X^1).2.(a, X^2)) = (3, a).0$. For a run ρ projecting onto a sequence μ, we write $Trace(\rho)$ for $Trace(\mu)$. The set of traces of runs of $\mathcal{A}$ is denoted by $\mathsf{Traces}(\mathcal{A}) \subseteq (\mathbb{R}_+ \cup \Sigma^{\mathcal{A}}_{obs})^*.\mathbb{R}_+$. Two OTAIOs are said *equivalent* if they have the same sets of traces.

Let $\sigma \in (\mathbb{R}_+ \cup \Sigma^{\mathcal{A}}_{obs})^*.\mathbb{R}_+$ be an observable timed word, and $s \in S^{\mathcal{A}}$ a state, $\mathcal{A}$ `after` $\sigma = \{s \in S^{\mathcal{A}} \mid \exists \mu \in \mathsf{Seq}(\mathcal{A}), s_0^{\mathcal{A}} \xrightarrow{\mu}_{\mathcal{A}} s \wedge trace(\mu) = \sigma\}$ denotes the set of states where $\mathcal{A}$ can stay after observing the trace σ. We note $elapse(s) = \{t \in \mathbb{R}_+ \mid s \xrightarrow{t}_{\mathcal{A}}\}$ the set of possible delays in s, and $out(s) = \{a \in \Sigma^{\mathcal{A}}_! \mid \exists X \subseteq X^{\mathcal{A}}, s \xrightarrow{(a,X)}_{\mathcal{A}}\} \cup elapse(s)$ (and $in(s) = \{a \in \Sigma^{\mathcal{A}}_? \mid s \xrightarrow{(a,X)}_{\mathcal{A}}\}$) for the set of outputs and delays (respectively inputs) that can be observed from s. For $S' \subset S^{\mathcal{A}}$, $out(S') = \bigcup_{s \in S'} out(s)$ and $in(S') = \bigcup_{s \in S'} in(s)$.

2.3 Properties and operations

An OTAIO $\mathcal{A}$ is *deterministic* (and called a DOTAIO) whenever it has no internal transition and for any $\sigma \in \mathsf{Traces}(\mathcal{A}), q_0^{\mathcal{A}}$ `after` σ is a singleton. Similarly a deterministic TAIO is called a DTAIO. An OTAIO $\mathcal{A}$ is *determinizable* if there exists an equivalent DOTAIO. 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 [16, 9].

An OTAIO $\mathcal{A}$ is *complete* if in every location ℓ, $I^{\mathcal{A}}(\ell) = \mathtt{true}$ 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 implies that $\mathsf{Traces}(\mathcal{A})$ is the universal language on $\Sigma^{\mathcal{A}}$. An OTAIO $\mathcal{A}$ is *input-complete* in a state $s \in reach(\mathcal{A})$, if for all $a \in \Sigma^{\mathcal{A}}_?$, $s \xrightarrow{a}$. An OTAIO $\mathcal{A}$ is *non-blocking* if $\forall s \in reach(\mathcal{A}), \forall t \in \mathbb{R}_+, \exists \mu \in \mathsf{pSeq}(\mathcal{A}) \cap (\mathbb{R}_+ \cup (\Sigma^{\mathcal{A}}_! \cup \Sigma^{\mathcal{A}}_\tau) \times 2^{X^{\mathcal{A}}})^*, time(\mu) = t \wedge s \xrightarrow{\mu}$.

We now define a product operation on OTAIOs which extends the classical product of TAs, with a particular attention to observed clocks:

Definition 2 (Product). *The* product *of two OTAIOs with same alphabets* $\mathcal{A}^i = (L^i, \ell_0^i, \Sigma_?, \Sigma_!, \Sigma_\tau, X_p^i, X_o^i, M^i, I^i, E^i)$, $i = 1, 2$, *and disjoint sets of proper clocks* ($X_p^1 \cap X_p^2 = \emptyset$) *is the OTAIO* $\mathcal{A}^1 \times \mathcal{A}^2 = (L, \ell_0, \Sigma_?, \Sigma_!, \Sigma_\tau, X_p, X_o, M, I, E)$ *where:* $L = L^1 \times L^2$; $\ell_0 = (\ell_0^1, \ell_0^2)$; $X_p = X_p^1 \cup X_p^2$, $X_o = (X_o^1 \cup X_o^2) \setminus X_p$; $M = max(M^1, M^2)$; $\forall (\ell^1, \ell^2) \in L, I((\ell^1, \ell^2)) = I^1(\ell^1) \wedge I^2(\ell^2)$; *and* $((\ell^1, \ell^2), g^1 \wedge g^2, a, X_p'^1 \cup X_p'^2, (\ell'^1, \ell'^2)) \in E$ *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 ones). $\mathcal{A}^2$ may observe proper clocks of $\mathcal{A}^1$ with its observed clocks $X_o^1 \cap X_p^2$, and vice versa. The set of proper clocks of $\mathcal{A}^1 \times A^2$ is the union of proper clocks of $\mathcal{A}^1$ and A^2, and observed clocks are those observed clocks of one OTAIO that are not proper. For example, the OTAIO in Fig. 3 represents the product of the TAIO $\mathcal{A}$ in Fig. 1 and the OTAIO $\mathcal{TP}$ of Fig. 2.

The product is the right operation for intersecting sets of sequences. In fact, let $\mathcal{A}^1{\uparrow}^{(X_p^2,X_o^2)}$ (respectively $\mathcal{A}^2{\uparrow}^{(X_p^1,X_o^1)}$) denote the same TAIO $\mathcal{A}^1$ (resp. $\mathcal{A}^2$) defined on $(X_p^1, X_p^2 \cup X_o^2 \cup X_o^1 \setminus X_p^1)$ (resp. on $(X_p^2, X_p^1 \cup X_o^2 \cup X_o^2 \setminus X_p^2)$). Then we get:

$$\mathsf{Seq}(\mathcal{A}^1 \times \mathcal{A}^2) = \mathsf{Seq}(\mathcal{A}^1{\uparrow}^{(X_p^2,X_o^2)}) \cap \mathsf{Seq}(\mathcal{A}^2{\uparrow}^{(X_p^1,X_o^1)}).$$

An OTAIO equipped with a set of states $F \subseteq S^{\mathcal{A}}$ can play the role of an acceptor. $\mathsf{Run}_F(\mathcal{A})$ denotes the set of runs *accepted* in F, those runs ending in F, $\mathsf{Seq}_F(\mathcal{A})$ denotes the set of sequences of accepted runs and $\mathsf{Traces}_F(\mathcal{A})$ the set of their traces. By abuse of notation, if L is a subset of locations $L^{\mathcal{A}}$, we write $\mathsf{Run}_L(\mathcal{A})$ for $\mathsf{Run}_{L \times \mathbb{R}_+^{X^{\mathcal{A}}}}(\mathcal{A})$ and similarly for $\mathsf{Seq}_L(\mathcal{A})$ and $\mathsf{Traces}_L(\mathcal{A})$. 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, we get the equality:

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

3 Conformance testing theory

In this section, we recall the conformance relation **tioco** [12], that formally defines the set of correct implementations of a given TAIO specification. **tioco** is a natural extension of the **ioco** relation of Tretmans [15] to timed systems. We then define test cases, formalize their executions, verdicts and expected properties. Finally, we introduce a refinement relation between TAIOs that preserves **tioco**.

3.1 The tioco conformance theory

We consider that the specification is given as a (possibly non-deterministic) TAIO $\mathcal{A} = (L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Sigma_?, \Sigma_!, \Sigma_\tau, X_p^{\mathcal{A}}, \emptyset, M^{\mathcal{A}}, I^{\mathcal{A}}, E^{\mathcal{A}})$. 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) TAIO $\mathcal{I} = (L^{\mathcal{I}}, \ell_0^{\mathcal{I}}, \Sigma_?, \Sigma_!, \Sigma_\tau^{\mathcal{I}}, X_p^{\mathcal{I}}, \emptyset, M^{\mathcal{I}}, I^{\mathcal{I}}, E^{\mathcal{I}})$ with same observable alphabet as $\mathcal{A}$, and require that it is input-complete and non-blocking. The set of such possible implementations of $\mathcal{A}$ is denoted by $\mathcal{I}(\mathcal{A})$. Among these, the conformance relation **tioco** [12] formally defines which ones conform to $\mathcal{A}$:

Definition 3 (Conformance relation). *Let $\mathcal{A}$ be a TAIO and $\mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\mathcal{I}$* **tioco** *$\mathcal{A}$ if $\forall \sigma \in \mathsf{Traces}(\mathcal{A}), out(\mathcal{I}\ \texttt{after}\ \sigma) \subseteq out(\mathcal{A}\ \texttt{after}\ \sigma)$.*

Intuitively, $\mathcal{I}$ conforms to $\mathcal{A}$ ($\mathcal{I}$ **tioco** $\mathcal{A}$) if after any timed trace enabled in $\mathcal{A}$, every output or delay of $\mathcal{I}$ is specified in $\mathcal{A}$. 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 4 (Test case, test suite). *Given a specification TAIO $\mathcal{A}$, a* test case *for $\mathcal{A}$ is a pair $(\mathcal{TC}, \textbf{Verdicts})$ consisting of a deterministic TAIO (DTAIO) $\mathcal{TC} = (L^{\mathcal{TC}}, \ell_0^{\mathcal{TC}}, \Sigma_?^{\mathcal{TC}}, \Sigma_!^{\mathcal{TC}}, \Sigma_\tau^{\mathcal{TC}}, X_p^{\mathcal{TC}}, \emptyset, M^{\mathcal{TC}}, I^{\mathcal{TC}}, E^{\mathcal{TC}})$ and a partition of the*

set of states $S^{\mathcal{TC}}$ *into* **Verdicts** = **None** ⊔ **Inconc** ⊔ **Pass** ⊔ **Fail** *where states outside* **None** *are called* verdict states. *We require that* $\Sigma_?^{\mathcal{TC}} = \Sigma_!^{\mathcal{A}}$ *and* $\Sigma_!^{\mathcal{TC}} = \Sigma_?^{\mathcal{A}}$, $I^{\mathcal{TC}}(\ell) = \texttt{true}$ *for all* $\ell \in L^{\mathcal{TC}}$, *and* $\mathcal{TC}$ *is input-complete in all* **None** *states, meaning that it is ready to receive any input from the implementation before reaching a verdict. A* test suite *is a set of test cases.*

We say that the verdict of an execution $\sigma \in \mathsf{Traces}(\mathcal{TC})$, noted $\texttt{Verdict}(\sigma, \mathcal{TC})$, is **Pass**, **Fail**, **Inconc** or **None** if $\mathcal{TC}$ $\texttt{after}\,\sigma$ is included in the corresponding states set. We note $\mathcal{I}$ $\texttt{fails}$ TC if some execution σ of $\mathcal{TC}\|\mathcal{I}$ leads $\mathcal{TC}$ to a **Fail** state, *i.e.* when $\mathsf{Traces}_{\mathbf{Fail}}(\mathcal{TC}) \cap \mathsf{Traces}(\mathcal{I}) \neq \emptyset$ [3]. Notice that this is only a possibility to reach the **Fail** verdict among the infinite set of executions.

Definition 5 (Test case properties). *A test suite* $\mathcal{TS}$ *for* $\mathcal{A}$ *is:*
- sound *if* $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\forall \mathcal{TC} \in \mathcal{TS}$, $\mathcal{I}$ $\texttt{fails}$ $\mathcal{TC} \Rightarrow \neg(\mathcal{I}$ **tioco** $\mathcal{A})$,
- strict *if* $\forall \mathcal{I} \in \mathcal{I}(A), \forall \mathcal{TC} \in \mathcal{TS}, \neg(\mathcal{I}\|\mathcal{TC}$ **tioco** $\mathcal{A}) \Rightarrow \mathcal{I}$ $\texttt{fails}$ $\mathcal{TC}$.

Soundness means that no conformant implementation is rejected by the test suite. This is a crucial property, ensured by our test generation method. In the other direction, strictness means that non-conformance is detected as soon as it occurs, and is ensured by our method when determinization is exact.

3.2 Refinement preserving tioco

We introduce an io-refinement relation between TAIOs, a generalization to non-deterministic TAIOs of the io-refinement between DTAIOs introduced in [6], itself a generalization of alternating simulation [2]. We 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 4.

Definition 6. *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*

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

It can be proved that $\preceq$ is a preorder relation. Moreover, as the second condition is always satisfied if $\mathcal{A}$ is input-complete, for $\mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\mathcal{I}$ **tioco** $\mathcal{A}$ is equivalent to $\mathcal{I} \preceq \mathcal{A}$. By transitivity of $\preceq$, it follows that io-refinement preserves conformance:

Proposition 1. *If* $\mathcal{A} \preceq \mathcal{B}$ *then* $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A})$ $(= \mathcal{I}(\mathcal{B}))$, $\mathcal{I}$ **tioco** $\mathcal{A} \Rightarrow \mathcal{I}$ **tioco** $\mathcal{B}$.

As a corollary, we get that io-abstraction preserves soundness of test suites:

Corollary 1. *If* $A \preceq B$ *then any sound test suite for* $\mathcal{B}$ *is also sound for* $\mathcal{A}$.

In the sequel, this corollary will justify that if an OTAIO $\mathcal{B}$ io-abstracting $\mathcal{A}$ can be obtained by approximate determinization, one can generate sound test cases for $\mathcal{B}$ that are still sound for $\mathcal{A}$. All proofs are given in the Appendix.

[3] The execution of a test case $\mathcal{TC}$ on an implementation $\mathcal{I}$ is usually modeled by the standard parallel composition $\mathcal{TC}\|\mathcal{I}$. Due to space limitations, $\|$ is not defined here, but we use its trace properties: $\mathsf{Traces}(\mathcal{I}\|\mathcal{TC}) = \mathsf{Traces}(\mathcal{I}) \cap \mathsf{Traces}(\mathcal{TC})$.

4 Approximate determinization preserving tioco

We recently proposed a game approach to determinize or provide a deterministic over-approximation for TAs [4]. Determinization is exact on all known classes of determinizable TAIOs (*e.g.* event-clock TAs, TAs with integer resets, strongly non-Zeno TAs) if resources are sufficient. Provided a couple of extensions, this method can be adapted to the context of testing for building a deterministic io-abstraction of a given TAIO. Thanks to Proposition 1, the construction preserves **tioco**, and thus guarantees the soundness of generated test cases.

The approximate determinization uses the classical region 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 like the reachability of a location. We note $\mathsf{Reg}_{(X,M)}$ the set of regions over clocks X with maximal constant M. A region r' is a *time-successor* of a region r if $\exists v \in r, \exists t \in \mathbb{R}_+, v + t \in r'$. Given X and Y two finite sets of clocks, a *relation* between clocks of X and those of Y is a finite conjunction C of atomic constraints of the form $x - y \sim c$ where $x \in X$, $y \in Y$, $\sim \in \{<, =, >\}$ and $c \in \mathbb{N}$. When $c \in [-M', M]$, for $M, M' \in \mathbb{N}$, we denote by $\mathsf{Rel}_{M,M'}(X, Y)$ the set of relations between X and Y.

4.1 A game approach to determinize timed automata

The technique presented in [4] applies first to TAs, *i.e.* the alphabet only consists of one kind of actions (output actions), and the invariants are all trivial. Given such a TA $\mathcal{A}$ over the set of clocks $X^{\mathcal{A}}$, the goal is to build a deterministic TA $\mathcal{B}$ with $\mathsf{Traces}(\mathcal{A}) = \mathsf{Traces}(\mathcal{B})$ as often as possible, or $\mathsf{Traces}(\mathcal{A}) \subseteq \mathsf{Traces}(\mathcal{B})$. In order to do so, resources of $\mathcal{B}$ (number of clocks k and maximal constant $M^{\mathcal{B}}$) are fixed, and a finite 2-player turn-based safety game $\mathcal{G}_{\mathcal{A},(k,M^{\mathcal{B}})}$ 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 $\mathsf{Traces}(\mathcal{A}) \subseteq \mathsf{Traces}(\mathcal{B})$, and every winning strategy induces a deterministic TA $\mathcal{B}$ equivalent to $\mathcal{A}$.

Let us now give more details on the definition of the game. Let $X^{\mathcal{B}}$ be a set of clocks of cardinality k. The initial state of the game is a state of Spoiler consisting of the initial location of $\mathcal{A}$, the simplest relation between $X^{\mathcal{A}}$ and $X^{\mathcal{B}}$: $\forall x \in X^{\mathcal{A}}, \forall y \in X^{\mathcal{B}}, x - y = 0$, a marking $\top$ indicating that no over-approximation was done so far, together with the null region over $X^{\mathcal{B}}$. In each of his states, Spoiler challenges Determinizator by proposing a region $r \in \mathsf{Reg}_{(X^{\mathcal{B}},M^{\mathcal{B}})}$, and an action $a \in \Sigma$. Determinizator answers by deciding the subset of clocks $Y' \subseteq X^{\mathcal{B}}$ he wishes to reset. The next state of Spoiler contains a region over $X^{\mathcal{B}}$ ($r' = r_{[Y' \leftarrow 0]}$), and a finite set of configurations: triples formed of a location of $\mathcal{A}$, a relation between clocks in $X^{\mathcal{A}}$ and clocks in $X^{\mathcal{B}}$, and a boolean marking ($\top$ or $\bot$). A state of Spoiler thus constitutes a states' estimate of $\mathcal{A}$, and the role of the markings is to indicate whether over-approximations possibly happened. Bad states Determinizator wants to avoid are states where all configurations are marked $\bot$, *i.e.* configurations where an approximation possibly happened.

A strategy for Determinizator thus assigns to each state of Determinizator a set $Y' \subseteq X^{\mathcal{B}}$ of clocks to be reset. With every strategy for Determinizator Π we associate the TA $\mathcal{B} = \mathsf{Aut}(\Pi)$ 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 1 ([4]). *Let $\mathcal{A}$ be a TA, and $k, M^{\mathcal{B}} \in \mathbb{N}$. For every strategy Π of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M^{\mathcal{B}})}$, $\mathcal{B} = \mathsf{Aut}(\Pi)$ is a deterministic timed automaton over resources $(k, M^{\mathcal{B}})$ and satisfies $\mathsf{Traces}(\mathcal{A}) \subseteq \mathsf{Traces}(\mathcal{B})$. Moreover, if Π is winning, then $\mathsf{Traces}(\mathcal{A}) = \mathsf{Traces}(\mathcal{B})$.*

4.2 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 [4], and summarized below. First the model of TAIOs is more expressive than TAs and incorporates internal actions and invariants. Second, in order to preserve **tioco**, the goal is to build from a TAIO $\mathcal{A}$ a DTAIO $\mathcal{B}$ such that $\mathcal{A} \preceq \mathcal{B}$, thus inputs and outputs must be treated differently.

Internal actions: Specifications naturally include internal actions that 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. To this attempt, states of the game have to be extended since internal actions might be enabled only from some time-successor 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 silent transitions is effectively computed the same way as successors in the original construction when Determinizator does not reset any clock, computations thus terminate for the same reasons. It is well known that timed automata with silent transitions are strictly more expressive than standard timed automata [3]. Therefore, our approximation can be coarse, but it performs as well as possible with its available clock information.

Invariants: Modeling urgency is quite important and using invariants to this aim is classical. Without the ability to express urgency, for instance, any inactive system would conform to all specifications. Ignoring all invariants in the approximation surely yields an io-abstraction: delays (considered as outputs) are over-approximated. In order to be more precise, while preserving $\preceq$, 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 extremities of the transition. A state whose invariant is strictly over-approximated is not safe.

io-abstraction vs. *over-approximation:* Rather than over-approximating a given TAIO $\mathcal{A}$, we aim here at building a DTAIO $\mathcal{B}$ such that $\mathcal{B}$ io-abstracts $\mathcal{A}$ ($\mathcal{A} \preceq \mathcal{B}$). Successors by output 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- and under-approximated closures. Thus, the litigious successors by an input (where possibly an over-approximation would be done), are not built.

All in all, these modifications allow to deal with the full TAIO model with invariants, silent transitions and inputs/outputs. In particular, the treatment of invariants is consistent with the io-abstraction: delays are considered as outputs, thus over-approximated. Fig.4 represents a part of this game for the TAIO of Fig.3. The new game then enjoys the following nice property:

Proposition 2 ([4] [4]). *Let $\mathcal{A}$ be a TAIO, and $k, M^{\mathcal{B}} \in \mathbb{N}$. For every strategy Π of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M^{\mathcal{B}})}$, $\mathcal{B} = \mathsf{Aut}(\Pi)$ is a DTAIO over resources $(k, M^{\mathcal{B}})$ with $\mathcal{A} \preceq \mathcal{B}$. Moreover, if Π is winning, then $\mathsf{Traces}(\mathcal{A}) = \mathsf{Traces}(\mathcal{B})$.*

In other words, the approximations produced by our method are deterministic io-abstractions of the initial specification, hence our approach preserves **tioco** (Proposition 1) and soundness of test cases (Corollary 1). In comparison, the algorithm proposed in [12] is an over-approximation, thus preserves **tioco** only if the specification is input-complete. Moreover it does not preserve urgency.

5 Off-line test case generation

In this section we first define test purposes and then give the principles for off-line test selection with test purposes and properties of generated test cases.

5.1 Test purposes

Test purposes are practical means to select behaviors to be tested, either focusing on usual behaviors, or on suspected errors in implementations. In this work we choose the following definition of test purposes, and discuss some alternatives in the conclusion.

Definition 7 (Test purpose). *For a specification TAIO $\mathcal{A}$, a* test purpose *is a pair $(\mathcal{TP}, \mathsf{Accept})$ where $\mathcal{TP} = (L^{\mathcal{TP}}, \ell_0^{\mathcal{TP}}, \Sigma_?, \Sigma_!, \Sigma_\tau, X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}}, M^{\mathcal{TP}}, I^{\mathcal{TP}}, E^{\mathcal{TP}})$ is a complete OTAIO (in particular $I^{\mathcal{TP}}(\ell) = \mathtt{true}$ 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 $\mathsf{Accept} \subseteq L^{\mathcal{TP}}$ is a subset of trap locations.*

Fig. 2 represents a test purpose for the specification $\mathcal{A}$ of Fig. 1. It 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) is an abbreviation for the complement of specified transitions.

[4] Note that the proof of this proposition in [4] considers a stronger refinement relation, thus implies the same result for the present refinement relation.

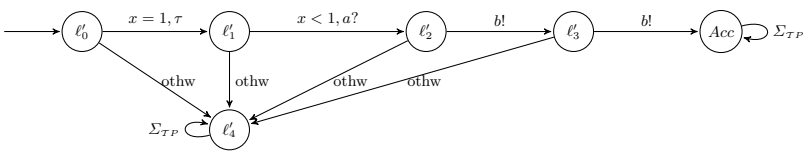

Fig. 2. Test purpose $\mathcal{TP}$.

5.2 Principle of test generation

Given a specification TAIO $\mathcal{A}$ and a test purpose $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$, our aim is to build a test case $(\mathcal{TC}, \mathbf{Verdicts})$ which is sound and, if possible, strict. It should also focus on traces of sequences accepted by $\mathcal{TP}$. This is formalized by the following property:

Definition 8. *A test suite $\mathcal{TS}$ for $\mathcal{A}$ and $\mathcal{TP}$ is* precise *if* $\forall \mathcal{TC} \in \mathcal{TS}, \forall \sigma \in (\Sigma^{\mathcal{A}}_{obs})^*, \mathtt{Verdict}(\sigma, \mathcal{TC}) = \mathbf{Pass} \iff \sigma \in \mathsf{Traces}(\mathsf{Seq}^{\mathcal{TP}}_{\mathsf{Accept}}(\mathcal{TP}) \cap \mathsf{Seq}(\mathcal{A}))$, *meaning that* **Pass** *verdicts are correctly delivered on traces of sequences of $\mathcal{A}$ accepted by $\mathcal{TP}$ in* $\mathsf{Accept}^{\mathcal{TP}}$.

The different steps of test generation are described in the following paragraphs.

Product: we first build the TAIO $\mathcal{P} = \mathcal{A} \times \mathcal{TP}$ associated with the set of marked locations $\mathsf{Accept}^{\mathcal{P}} = L^{\mathcal{A}} \times \mathsf{Accept}^{\mathcal{TP}}$. Fig. 3 represents this product $\mathcal{P}$ for the specification $\mathcal{A}$ in Fig. 1 and the test purpose $\mathcal{TP}$ in Fig. 2. The effect of the product is to unfold $\mathcal{A}$ and to mark those sequences of $\mathcal{A}$ accepted by $\mathcal{TP}$ in locations $\mathsf{Accept}^{\mathcal{TP}}$. $\mathcal{TP}$ is complete, thus $\mathsf{Seq}(\mathcal{P}) = \mathsf{Seq}(\mathcal{A}{\uparrow}^{X_p^{\mathcal{TP}}, X_o^{\mathcal{TP}}})$ (sequences of the product are sequences of $\mathcal{A}$ lifted to $X^{\mathcal{TP}}$), and then $\mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{A})$, which implies that $\mathcal{P}$ and $\mathcal{A}$ define the same sets of conformant implementations. We also have $\mathsf{Seq}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}) = \mathsf{Seq}(\mathcal{A}{\uparrow}^{X_p^{\mathcal{TP}}; X_o^{\mathcal{TP}}}) \cap \mathsf{Seq}_{\mathsf{Accept}^{\mathcal{TP}}}(\mathcal{TP})$ which induces $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P}) = \mathsf{Traces}(\mathsf{Seq}(\mathcal{A}) \cap \mathsf{Seq}_{\mathsf{Accept}^{\mathcal{TP}}}(\mathcal{TP}))$.

Let $\mathsf{ATraces}(\mathcal{A}, \mathcal{TP}) = \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{P}}}(\mathcal{P})$ and $\mathsf{RTraces}(\mathcal{A}, \mathcal{TP}) = \mathsf{Traces}(\mathcal{A}) \setminus \mathit{pref}(\mathsf{ATraces}(\mathcal{A}, \mathcal{TP}))$ where, for a set of traces T, $\mathit{pref}(T)$ denotes the set of prefixes of traces in T. The principle is to select traces in $\mathsf{ATraces}(\mathcal{A}, \mathcal{TP})$ and try to avoid or at least detect those in $\mathsf{RTraces}(\mathcal{A}, \mathcal{TP})$ as these traces cannot be prefixes of traces of sequences satisfying the test purpose.

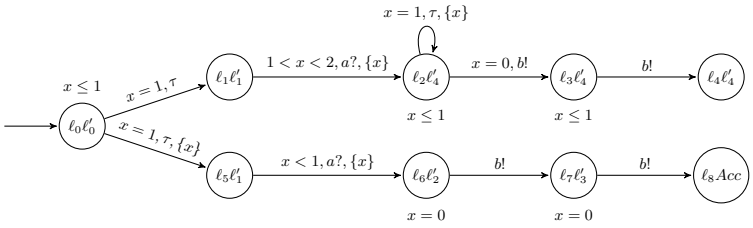

Fig. 3. Product $\mathcal{P} = \mathcal{A} \times \mathcal{TP}$.

Approximate determinization of $\mathcal{P}$ into $\mathcal{DP}$: If $\mathcal{P}$ is already deterministic, we simply take $\mathcal{DP} = \mathcal{P}$. Otherwise, with the approximate determinization of Section 4, we can build a deterministic io-abstraction $\mathcal{DP}$ of $\mathcal{P}$ with resources $(k, M^{\mathcal{DP}})$ fixed by the user, thus $\mathcal{P} \preceq \mathcal{DP}$. $\mathcal{DP}$ is equipped with the set of

marked locations $\mathsf{Accept}^{\mathcal{DP}}$ consisting of locations in $L^{\mathcal{DP}}$ containing some configuration which location is in $\mathsf{Accept}^{\mathcal{P}}$. If the determinization is exact, we get $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$ and $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{DP}}}(\mathcal{DP}) = \mathsf{ATraces}(\mathcal{A}, \mathcal{TP})$. Fig. 4 partially represents the game $\mathcal{G}_{\mathcal{P},(1,2)}$ for the TAIO $\mathcal{P}$ of Fig. 3 where, for readability reasons, some behaviors not co-reachable from $\mathsf{Accept}^{\mathcal{DP}}$ are omitted. $\mathcal{DP}$ is simply obtained from $\mathcal{G}_{\mathcal{P},(1,2)}$ by merging transitions of Spoiler and Determinizator.

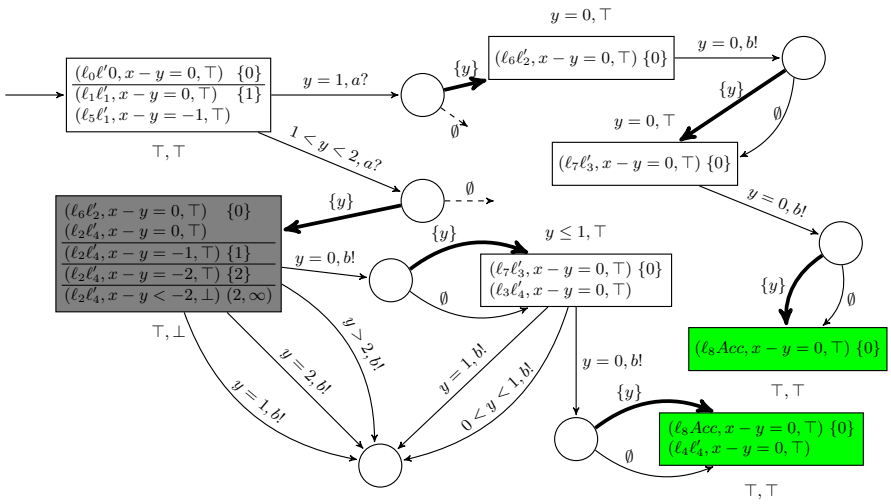

Fig. 4. Game $\mathcal{G}_{\mathcal{P},(1,2)}$.

Generating $\mathcal{TC}$ from $\mathcal{DP}$: The next step consists in building $(\mathcal{TC}, \mathbf{Verdicts})$ from $\mathcal{DP}$, using an analysis of the co-reachability to locations $\mathsf{Accept}^{\mathcal{DP}}$ in $\mathcal{DP}$.

The test case built from $\mathcal{DP} = (L^{\mathcal{DP}}, \ell_0^{\mathcal{DP}}, \Sigma_?^{\mathcal{DP}}, \Sigma_!^{\mathcal{DP}}, X_p^{\mathcal{DP}}, \emptyset, M^{\mathcal{DP}}, I^{\mathcal{DP}}, E^{\mathcal{DP}})$ and $\mathsf{Accept}^{\mathcal{DP}}$ is the TAIO $\mathcal{TC} = (L^{\mathcal{TC}}, \ell_0^{\mathcal{TC}}, \Sigma_?^{\mathcal{TC}}, \Sigma_!^{\mathcal{TC}}, X_p^{\mathcal{TC}}, \emptyset, M^{\mathcal{TC}}, I^{\mathcal{TC}}, E^{\mathcal{TC}})$ such that $L^{\mathcal{TC}} = L^{\mathcal{DP}} \sqcup \{\ell_{\mathbf{Fail}}\}$ where $\ell_{\mathbf{Fail}}$ is a new location; $\ell_0^{\mathcal{TC}} = \ell_0^{\mathcal{DP}}$; $\Sigma_?^{\mathcal{TC}} = \Sigma_!^{\mathcal{DP}} = \Sigma_!^{\mathcal{A}}$ and $\Sigma_!^{\mathcal{TC}} = \Sigma_?^{\mathcal{DP}} = \Sigma_?^{\mathcal{A}}$, *i.e.* input/output alphabets are mirrored in order to reflect the opposite role of actions in the synchronization of $\mathcal{TC}$ and $\mathcal{I}$; $X_p^{\mathcal{TC}} = X_p^{\mathcal{DP}}$ and $X_o^{\mathcal{TC}} = \emptyset$; $M^{\mathcal{TC}} = M^{\mathcal{DP}}$; **Verdicts** is the partition of $S^{\mathcal{TC}}$ with $\mathbf{Pass} = \bigcup_{\ell \in \mathsf{Accept}^{\mathcal{DP}}} \{\ell\} \times I^{\mathcal{DP}}(\ell)$, $\mathbf{None} = \mathtt{coreach}(\mathcal{DP}, \mathbf{Pass}) \setminus \mathbf{Pass}$, $\mathbf{Inconc} = S^{\mathcal{DP}} \setminus \mathtt{coreach}(\mathcal{DP}, \mathbf{Pass})$, and $\mathbf{Fail} = \{\ell_{\mathbf{Fail}}\} \times \mathbb{R}_+^{X^{\mathcal{TC}}} \cup \{(\ell, \neg I^{\mathcal{DP}}(\ell)) | \ell \in L^{\mathcal{DP}}\}$; $I^{\mathcal{TC}}(\ell) = \mathtt{true}$ for any $\ell \in L^{\mathcal{TC}}$; $E^{\mathcal{TC}} = E_I^{\mathcal{DP}} \cup E_{\ell_{\mathbf{Fail}}}$ where $E_I^{\mathcal{DP}} = \{(\ell, g \wedge I^{\mathcal{DP}}(\ell), a, X, \ell') \mid (\ell, g, a, X, \ell') \in E^{\mathcal{DP}}\}$ and $E_{\ell_{\mathbf{Fail}}} = \{(\ell, \bar{g}, a, X_p^{\mathcal{TC}}, \ell_{\mathbf{Fail}}) \mid \ell \in L^{\mathcal{DP}}, a \in \Sigma_!^{\mathcal{DP}}, \bar{g} = \neg \bigvee_{(\ell,g,a,X,\ell') \in E^{\mathcal{DP}}} g\}$.

The important points to understand in the construction of $\mathcal{TC}$ are the completion to **Fail** and the computation of **Inconc**. For the completion, the idea is to detect unspecified outputs and delays of $\mathcal{DP}$. Outputs of $\mathcal{DP}$ being inputs of $\mathcal{TC}$, in any location ℓ, for each input $a \in \Sigma_?^{\mathcal{TC}} = \Sigma_!^{\mathcal{DP}}$, a transition leading to $\ell_{\mathbf{Fail}}$ is added, labeled with a, and which guard is the negation of the disjunction of all guards of transitions labeled by a and leaving ℓ (thus `true` if no a-action leaves ℓ). Authorized delays in $\mathcal{DP}$ being defined by invariants, all states in $(\ell, \neg I^{\mathcal{DP}}(\ell)), \ell \in L^{\mathcal{DP}}$, *i.e.* states where the invariant runs out, are put into **Fail**.

Moreover, in each location ℓ, the invariant $I^{\mathcal{DP}}(\ell)$ in $\mathcal{DP}$ is removed and shifted to guards of all transitions leaving ℓ in $\mathcal{TC}$.

The computation of **Inconc** is based on an analysis of the co-reachability to **Pass**. **Inconc** contains all states not co-reachable from locations in **Pass**. Notice that $\texttt{coreach}(\mathcal{DP}, \textbf{Pass})$, and thus **Inconc**, can be computed symbolically in the region graph of $\mathcal{DP}$. Fig.5 represents the test case obtained from $\mathcal{A}$ and $\mathcal{TP}$.

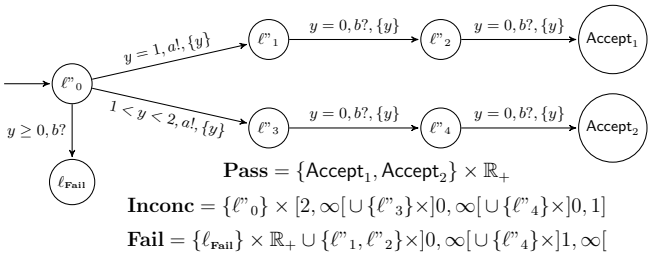

Fig. 5. Test case $\mathcal{TC}$

Test selection: So far, the construction of $\mathcal{TC}$ determines **Verdicts**, but does not perform any selection of behaviors. A last step consists in trying to control the behavior of $\mathcal{TC}$ in order to avoid **Inconc** states (thus stay in $\mathit{pref}(\mathsf{ATraces}(\mathcal{A}, \mathcal{TP}))$), or produce an **Inconc** verdict when this is impossible. To this aim, guards of transitions are refined in two complementary ways. First, transitions leaving a verdict state are useless, thus for each transition, the guard is intersected with the set of valuations associated with **None** in the source location. Second, transitions arriving in **Inconc** states and carrying inputs are also useless, thus for any transition labeled by an input, the guard is intersected with the set of valuations associated with $\texttt{coreach}(\mathcal{DP}, \textbf{Pass})$ in the target location. For example in $\mathcal{TC}$ (Fig. 5), the bottom-left state of the game in Fig. 4 has been removed.

After these steps, generated test cases exhibit the following properties:

Theorem 2. *Any test case $\mathcal{TC}$ built by the procedure is sound for $\mathcal{A}$. If $\mathcal{DP}$ is an exact approximation of $\mathcal{P}$, $\mathcal{TC}$ is also strict and precise for $\mathcal{A}$ and $\mathcal{TP}$.*

The proof is given in the Appendix. Soundness comes from the construction of $E_{\textbf{Fail}}$ in $\mathcal{TC}$ and preservation of soundness by the approximate determinization $\mathcal{DP}$ of $\mathcal{P}$ given by Corollary 1. When $\mathcal{DP}$ is an exact determinization of $\mathcal{P}$, $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{A})$. Strictness then comes from the fact that $\mathcal{DP}$ and $\mathcal{A}$ have the same non-conformant traces and from the definition of $E_{\textbf{Fail}}$ in $\mathcal{TC}$. Precision comes from $\mathsf{Traces}_{\mathsf{Accept}^{\mathcal{DP}}}(\mathcal{DP}) = \mathsf{ATraces}(\mathcal{A}, \mathcal{TP})$ 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 non-conformant behaviors.

Test execution After test selection, it remains to execute test cases on a real implementation. As the test case is a TAIO, a number of decisions still need to be made at each node of the test case: (1) whether to wait for a certain delay, to receive an input or emit an output (2) which output to send, in case there is a choice. Some of these choices can be made either randomly, or according to user-defined strategies, for example by applying a technique similar to the control approach of [7] which goal is to avoid $\mathsf{RTraces}(\mathcal{A}, \mathcal{TP})$.

6 Conclusion

In this paper, 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 TAIO, thanks to an original approximate determinization procedure. Another main contribution is the selection of test cases with expressive test purposes described as OTAIOs having the ability to precisely describe 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.

Related work and discussion: As mentioned in the introduction, off-line test selection is in general limited to deterministic or determinizable timed automata, except in [12] which relies on an approximate determinization. Compared to this work, our approximate determinization is more precise (it is exact in more cases) and preserves urgency in test cases as much as possible.

In several other works [11, 8], test purposes are used for test case selection from TAIOs. In all these works, test purposes only have proper clocks, but cannot observe clocks of the specification. The advantage of our definition is its generality and a fine tuning of selection. One could argue that the cost of producing a test suite can be heavy, as for each test purpose, the whole sequence of operations, including determinization, must be done. In order to avoid this, an alternative would be to define test purposes recognizing timed traces and perform selection on the approximate determinization $\mathcal{B}$ of $\mathcal{A}$. But then, the test purpose should not use $\mathcal{A}$'s clocks as these are lost by determinization. Then, test purposes are either defined after determinization and observe $\mathcal{B}$'s clocks, or their expressive power is further restricted by using only proper clocks in order not to depend on $\mathcal{B}$.

Concerning test selection, in [7], the authors propose a game approach which effect can be understood as a way to completely avoid $\mathsf{RTraces}(\mathcal{A}, \mathcal{TP})$, with the possible risk to miss some or even all traces in $\mathit{pref}(\mathsf{ATraces}(\mathcal{A}, \mathcal{TP}))$. Our selection, which allows to lose the game and produce an **Inconc** verdict when this happens, is both more liberal and closer to usual practice.

It should be noticed that selection by test purposes can be used for test selection with respect to coverage criteria. 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.

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. 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.
4. 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.
5. L. B. Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In *FATES'2004*, volume 3395 of *LNCS*, pages 64–78, 2005.
6. 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, 2010.
7. A. David, K. G. Larsen, S. Li, and B. Nielsen. Timed testing under partial observability. In *International Conference on Software Testing Verification and Validation (ICST'09)*, pages 61 –70, 2009.
8. A. En-Nouaary and R. Dssouli. A guided method for testing timed input output automata. In *Testing of Communicating Systems (TestCom'03)*, volume 2644 of *LNCS*, pages 211–225, 2003.
9. 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.
10. 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.
11. O. Koné, R. Castanet, and P. Laurencot. On the fly test generation for real time protocols. In *International Conference on Computer Communications & Networks (IC3N'98)*, 1998.
12. M. Krichen and S. Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, 2009.
13. B. Nielsen and A. Skou. Automated test generation from timed automata. *STTT*, 5:59–77, 2003.
14. 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, 2008.
15. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 3:103–120, 1996.
16. S. Tripakis. Folk theorems on the determinization and minimization of timed automata. *Information Processing Letters*, 99(6):222–226, 2006.

Appendix

Proof of Proposition 1 and Corollary 1

We start by proving that $\preceq$ is a preorder. It 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:

$$\forall \sigma \in \mathsf{Traces}(\mathcal{B}), out(\mathcal{A} \texttt{ after } \sigma) \subseteq out(\mathcal{B} \texttt{ after } \sigma) \quad (1)$$
$$\forall \sigma \in \mathsf{Traces}(\mathcal{A}), in(\mathcal{B} \texttt{ after } \sigma) \subseteq in(\mathcal{A} \texttt{ after } \sigma) \quad (2) \quad \textit{and}$$
$$\forall \sigma \in \mathsf{Traces}(\mathcal{C}), out(\mathcal{B} \texttt{ after } \sigma) \subseteq out(\mathcal{C} \texttt{ after } \sigma) \quad (3)$$
$$\forall \sigma \in \mathsf{Traces}(\mathcal{B}), in(\mathcal{C} \texttt{ after } \sigma) \subseteq in(\mathcal{B} \texttt{ after } \sigma) \quad (4)$$

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

$$\forall \sigma \in \mathsf{Traces}(\mathcal{C}), out(\mathcal{A} \texttt{ after } \sigma) \subseteq out(\mathcal{C} \texttt{ after } \sigma) \quad (5)$$
$$\forall \sigma \in \mathsf{Traces}(\mathcal{A}), in(\mathcal{C} \texttt{ after } \sigma) \subseteq in(\mathcal{A} \texttt{ after } \sigma) \quad (6)$$

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

- If $\sigma \in \mathsf{Traces}(\mathcal{B}) \cap \mathsf{Traces}(\mathcal{C})$ then by (1) and (3) we get $out(\mathcal{A} \texttt{ after } \sigma) \subseteq out(\mathcal{B} \texttt{ after } \sigma)$ and $out(\mathcal{B} \texttt{ after } \sigma) \subseteq out(\mathcal{C} \texttt{ after } \sigma)$ thus $out(\mathcal{A} \texttt{ after } \sigma) \subseteq out(\mathcal{C} \texttt{ after } \sigma)$ and we are done.
- If $\sigma \in \mathsf{Traces}(\mathcal{C}) \setminus \mathsf{Traces}(\mathcal{B})$, there exists σ', σ'' and $a \in \Sigma_{obs}$ such that $\sigma = \sigma'.a.\sigma''$ with $\sigma' \in \mathsf{Traces}(B)$ and $\sigma'.a \in \mathsf{Traces}(\mathcal{C}) \setminus \mathsf{Traces}(\mathcal{B})$. As $\mathcal{B} \preceq C$, by (4) we get that $a \in \Sigma_! \cup \mathbb{R}_+$. But as $\mathcal{A} \preceq \mathcal{B}$, and $\sigma' \in \mathsf{Traces}(\mathcal{B})$, condition (1) induces $out(\mathcal{A} \texttt{ after } \sigma') \subseteq out(\mathcal{B} \texttt{ after } \sigma')$, and then $\sigma'.a \in \mathsf{Traces}(\mathcal{C}) \setminus \mathsf{Traces}(\mathcal{A})$. Thus $out(\mathcal{A} \texttt{ after } \sigma'.a) = \emptyset$ and we conclude by $out(\mathcal{A} \texttt{ after } \sigma) = \emptyset \subseteq out(\mathcal{C} \texttt{ after } \sigma)$.

The proof of (6) is almost symmetric.

We now prove Proposition 1:

Proposition 1. If $\mathcal{A} \preceq \mathcal{B}$ then $\forall \mathcal{I} \in \mathcal{I}(\mathcal{A}) = \mathcal{I}(\mathcal{B})$, $\mathcal{I}$ **tioco** $\mathcal{A} \Rightarrow \mathcal{I}$ **tioco** $\mathcal{B}$.

The proof is then a direct consequence of the transitivity of $\preceq$. In fact when $\mathcal{I}$ is input complete, $\forall \sigma \in \mathsf{Traces}(\mathcal{I}), in(\mathcal{A} \texttt{ after } \sigma) \subseteq in(\mathcal{I} \texttt{ after } \sigma) = \Sigma_!$ trivially holds. Thus $\mathcal{I}$ **tioco** $\mathcal{A}$ (which is defined by $\forall \sigma \in \mathsf{Traces}(\mathcal{A}), out(\mathcal{I} \texttt{ after } \sigma) \subseteq out(\mathcal{A} \texttt{ after } \sigma)$) is equivalent to $\mathcal{I} \preceq \mathcal{A}$. Now suppose $\mathcal{A} \preceq B$ and $\mathcal{I}$ **tioco** $\mathcal{A}$ then the transitivity of $\preceq$ gives $\mathcal{I}$ **tioco** $\mathcal{B}$.

Remark: unfortunately, the converse of Proposition 1 is in general false. This comes from the fact that when a specification does not specify an input after a trace, for conformance this is equivalent to specifying this input and then accept the universal language on Σ_{obs}. A counter-example of the converse of Proposition 1 then consists in taking $\mathcal{A}$ which receives no input, and B receiving an input a and then accepting Σ_{obs}. They have same sets of conformant implementations, but $\neg(\mathcal{A} \preceq \mathcal{B})$ as $in(\mathcal{B} \texttt{ after } \epsilon) = a \not\subseteq in(\mathcal{A} \texttt{ after } \epsilon) = \emptyset$.

We now prove Corollary 1:
Corollary 1. If $A \preceq B$ then any sound test suite for $\mathcal{B}$ is also sound for $\mathcal{A}$.

Let $\mathcal{TS}$ be a sound test suite for $\mathcal{B}$. By definition $\forall \mathcal{I} \in \mathcal{I}(\mathcal{B})$, $\forall \mathcal{TC} \in \mathcal{TS}$, $\mathcal{I}$ `fails` $\mathcal{TC} \Rightarrow \neg(\mathcal{I}$ **tioco** $\mathcal{B})$. As we have $\mathcal{A} \preceq \mathcal{B}$, by Proposition 1, we have $\neg(\mathcal{I}$ **tioco** $\mathcal{B}) \Rightarrow \neg(\mathcal{I}$ **tioco** $\mathcal{A})$ which implies $\forall \mathcal{I} \in \mathcal{I}(\mathcal{B})$, $\forall \mathcal{TC} \in \mathcal{TS}$, $\mathcal{I}$ `fails` $\mathcal{TC} \Rightarrow \neg(\mathcal{I}$ **tioco** $\mathcal{B})$. Thus $\mathcal{TS}$ is sound for $\mathcal{A}$.

Proof of Theorem 2

Theorem 2. *Any test case $\mathcal{TC}$ built by the procedure is sound for $\mathcal{A}$. If $\mathcal{DP}$ is an exact approximation of $\mathcal{P}$, $\mathcal{TC}$ is also strict and precise for $\mathcal{A}$ and $\mathcal{TP}$.*

Soundness: To prove soundness, we need to show that for any $\mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\mathcal{I}$ `fails` $\mathcal{TC}$ implies $\neg(\mathcal{I}$ **tioco** $\mathcal{A})$. Assume that $\mathcal{I}$ `fails` $\mathcal{TC}$, then there exists a trace $\sigma \in \mathsf{Traces}(\mathcal{I}) \cap \mathsf{Traces}(\mathcal{TC})$ leading to **Fail**. By the construction of the set $E_{\mathbf{Fail}}$ in $\mathcal{TC}$, either $\sigma = \sigma'.a.0$ where $\sigma' \in \mathsf{Traces}(\mathcal{DP})$, $a \in \Sigma_!^{\mathcal{DP}}$ is unspecified in $\mathcal{DP}$ `after` σ', or $\sigma = \sigma'.\delta$ where $\delta > 0$ is unspecified in $\mathcal{DP}$ `after` σ. In both cases, this means that $\neg(\mathcal{I}$ **tioco** $\mathcal{DP})$, thus $\mathcal{TC}$ is sound for $\mathcal{DP}$. Now, as $\mathcal{DP}$ is an io-abstraction of $AxTP$ ($\mathcal{P} \preceq \mathcal{DP}$), by Corollary 1 this implies $\mathcal{TC}$ is sound for $\mathcal{P}$. Finally, we have $\mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{A})$, which trivially implies $\mathcal{A} \preceq \mathcal{P}$, and then $\mathcal{TC}$ is also sound for $\mathcal{A}$.

Strictness: For strictness, we have to prove that for any $\mathcal{I} \in \mathcal{I}(\mathcal{A})$, $\neg(\mathcal{I} \| \mathcal{TC}$ **tioco** $\mathcal{A})$ implies that $\mathcal{I}$ `fails` $\mathcal{TC}$. Suppose that $\neg(\mathcal{I} \| \mathcal{TC}$ **tioco** $\mathcal{A})$, then there exists $\sigma \in \mathsf{Traces}(\mathcal{A})$ and $a \in out(\mathcal{I} \| \mathcal{TC}$ `after` $\sigma)$ such that $a \notin out(\mathcal{A}$ `after` $\sigma)$. If $\mathcal{DP}$ is an exact approximation of $\mathcal{P}$, then $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{A})$, thus $\sigma \in \mathsf{Traces}(\mathcal{DP})$ and $a \notin out(\mathcal{DP}$ `after` $\sigma)$. By construction of $\mathcal{TC}$, it follows that $\sigma.a \in \mathsf{Traces}_{\mathbf{Fail}}(TC)$ which, together with $\sigma.a \in \mathsf{Traces}(\mathcal{I})$, implies that $\mathcal{I}$ `fails` $\mathcal{TC}$. Thus $\mathcal{TC}$ is strict.

Precision: To prove precision, we have to show that for any $\sigma \in (\Sigma_{obs}^{\mathcal{A}})^*$, $\mathtt{Verdict}(\sigma, \mathcal{TC}) = \mathbf{Pass} \iff \sigma \in \mathsf{Traces}(\mathsf{Seq}^{\mathcal{TP}}_{\mathsf{Accept}}(\mathcal{TP}) \cap \mathsf{Seq}(\mathcal{A}))$. The definition of $Pass = \bigcup_{\ell \in \mathsf{Accept}^{\mathcal{DP}}} \ell \times I^{\mathcal{DP}}(\ell)$ in $\mathcal{TC}$ implies that a **Pass** verdict is produced for σ exactly when $\sigma \in \mathsf{Traces}_{\mathsf{Accept}^{\mathcal{DP}}}(\mathcal{DP})$ which equals $\mathsf{ATraces}(\mathcal{A}, \mathcal{TP})$ and thus $\mathsf{Traces}(\mathsf{Seq}^{\mathcal{TP}}_{\mathsf{Accept}}(\mathcal{TP}) \cap \mathsf{Seq}(\mathcal{A}))$ when $\mathcal{DP}$ is exact.