Control strategies for off-line testing of timed systems

Léo Henry, Thierry Jéron, Nicolas Markey

Control strategies for off-line testing of timed systems

Léo Henry, Thierry Jéron, and Nicolas Markey

Univ. Rennes, INRIA & CNRS, Rennes (France)

Abstract. Partial observability and controllability are two well-known issues in test-case synthesis for interactive systems. We address the problem of partial control in the synthesis of test cases from timed-automata specifications. Building on the tioco timed testing framework, we extend a previous game interpretation of the test-synthesis problem from the untimed to the timed setting. This extension requires a deep reworking of the models, game interpretation and test-synthesis algorithms. We exhibit strategies of a game that tries to minimize both control losses and distance to the satisfaction of a test purpose, and prove they are winning under some fairness assumptions. This entails that when turning those strategies into test cases, we get properties such as soundness and exhaustiveness of the test synthesis method.

1 Introduction

Real-time interactive systems are systems interacting with their environment and subject to timing constraints. Such systems are encountered in many contexts, in particular in critical applications such as transportation, control of manufacturing systems, etc. Their correctness is then of prime importance, but it is also very challenging due to multiple factors: combination of discrete and continuous behaviours, concurrency aspects in distributed systems, limited observability of behaviours, or partial controllability of systems.

One of the most-used validation techniques in this context is testing, with variations depending on the design phases. Conformance testing is one of those variations, consisting in checking whether a real system correctly implements its specification. Those real systems are considered as black boxes, thereby offering only partial observability, for various reasons (*e.g.* because sensors cannot observe all actions, or because the system is composed of communicating components whose communications cannot be all observed, or again because of intellectual property of peer software). Controllability is another issue when the system makes its own choices upon which the environment, and thus the tester, have a limited control. One of the most-challenging activities in this context is the design of test cases that, when executed on the real system, should produce meaningful information about the conformance of the system at hand with respect to its specification. Formal models and methods are a good candidate to help this test-case synthesis [Tre96].

Timed Automata (TA) [AD94] form a class of model for the specification of timed reactive systems. It consists of automata equipped with real-valued clocks where transitions between locations carry actions, are guarded by constraints on clock values, and can reset clocks. TAs are also equipped with invariants that constrain the sojourn time in locations. TAs are popular in particular because reachability of a location is decidable using symbolic representations of sets of configurations by zones. In the context of testing, it is adequate to refine TAs by explicitly distinguishing (controllable) inputs and (uncontrollable) outputs, giving rise to TAIOs (Timed Automata with Inputs and Outputs). In the following, this model will be used for most testing artifacts, namely specifications, implementations, and test cases. Since completeness of testing is hopeless in practice, it is helpful to rely on test purposes that describe those behaviours that need to be tested because they are subject to errors. In our formal testing framework, an extension of TAIOs called Open TAIOs (or OTAIOs) is used to formally specify those behaviors. OTAIOs play the role of observers of actions and clocks of the specification: they synchronize on actions and clock resets of the specification (called observed clocks), and control their proper clocks. The formal testing framework also requires to formally define conformance as a relation between specifications and their possible implementations. In the timed setting, the classical tioco relation [KT09] states that, after a timed observable trace of the specification, the outputs and delays of the implementation should be specified.

Test-case synthesis from TAs has been extensively studied during the last 20 years (see [COG98,CKL98,SVD01,ENDK02,NS03,BB04,LMN04,KT09], to cite a few). As already mentioned, one of the difficulties is partial observation. In off-line testing, where the test cases are first computed, stored, and later executed on the implementation, the tester should anticipate all specified outputs after a trace. In the untimed framework, this is tackled by determinization of the specification. Unfortunately, this is not feasible for TAIO specifications since determinization is not possible in general [AD94,Fin06]. The solution was then either to perform on-line testing where subset construction is made on the current execution trace, or to restrict to determinizable sub-classes. More recently, some advances were obtained in this context [BJSK12] by the use of an approximate determinization using a game approach [BSJK15] that preserves tioco conformance. Partial observation is also dealt with by [DLL+10] with a variant of the TA model where observations are described by observation predicates composed of a set of locations together with clock constraints. Test cases are then synthesized as winning strategies, if they exist, of a game between the specification and its environment that tries to guide the system to satisfy the test purpose.

The problem of test synthesis is often informally presented as a game between the environment and the system (see *e.g.* [Yan04]). But very few papers effectively take into account the controllability of the system. In the context of testing for timed-automata models [DLLN08b], proposes a game approach where test cases are winning strategies of a reachability game. But this is restricted to deterministic models and controllability is not really taken into account. In fact, like in [DLL+10], the game is abandonned when control is lost, and it is suggested

to modify the test purpose in this case. This is mitigated in [DLLN08a] with cooperative strategies, which rely on the cooperation of the system under test to win the game. A more convincing approach to the control problem is the one of [Ram98] in the untimed setting, unfortunately a quite little-known work. The game problem consists in satisfying the test purpose (a simple sub-sequence), while trying to avoid control losses occurring when outputs offered by the system leave this behaviour. The computed strategy is based on a rank that measures both the distance to the goal and the controls losses.

The current paper adapts the approach proposed in [Ram98] to the timed context using the framework developed in [BJSK12]. Compared to [Ram98], the model of TA is much more complex than transition systems, the test purposes are also much more powerful than simple sub-sequences, thus even if the approach is similar, the game has to be completely revised. Compared to [DLL$^+$10], our model is a bit different since we do not rely on observation predicates, but partial observation comes from internal actions and choices. We do not completely tackle non-determinism since we assume determinizable models at some point. In comparison, [DLL$^+$10] avoids determinizing TAs, relying on the determinization of a finite state model, thanks to a projection on a finite set of observable predicates. Cooperative strategies of [DLLN08a] have similarities with our fairness assumptions, but their models are assumed deterministic. Our approach takes controllability into account in a more complete and practical way with the reachability game and rank-lowering strategies.

The paper is organized as follows. Chapter 2 introduces basic models: TAs, TAIOs and their open counterparts OTAs, OTAIOs, and then timed game automata (TGA). Chapter 3 is dedicated to the testing framework with hypothesis on models of testing artifacts, the conformance relation and the construction of the *objective-centered tester* that denotes both non-conformant traces and the goal to reach according to a test purpose. Chapter 4 constitutes the core of the paper. The test synthesis problem is interpreted as a game on the objective-centered tester. Rank-lowering strategies are proposed as candidate test cases, and a fairness assumption is introduced to make such strategies win. Finally properties of test cases with respect to conformance are proved.

2 Timed automata and timed games

In this section, we introduce our models for timed systems and for concurrent games on these objects, along with some useful notions and operations.

2.1 Timed automata with inputs and outputs

Timed automata (TAs) [AD94] are one of the most widely-used classes of models for reasoning about computer systems subject to real-time constraints. Timed automata are finite-state automata augmented with real-valued variables (called *clocks*) to constrain the occurrence of transitions along executions. In order to adapt these models to the testing framework, we consider TAs with inputs

and outputs (TAIOs), in which the alphabet is split between input, output and internal actions (the latter being used to model partial observation). We present the *open* TAs (and open TAIOs) [BJSK12], which allow the models to observe and synchronize with a set of non-controlled clocks.

Given a finite set of *clocks* X, a *clock valuation* over X is a function $v\colon X \to \mathbb{R}_{\geq 0}$. We note $\overline{0}_X$ (and often omit to mention X when clear from the context) for the valuation assigning 0 to all clocks in X. Let v be a clock valuation, for any $t \in \mathbb{R}_{\geq 0}$, we denote with $v + t$ the valuation mapping each clock $x \in X$ to $v(x) + t$, and for a subset $X' \subseteq X$, we write $v_{[X' \leftarrow 0]}$ for the valuation mapping all clocks in X' to 0, and all clocks in $X \setminus X'$ to their values in v.

A *clock constraint* is a finite conjunction of atomic constraints of the form $x \sim n$ where $x \in X$, $n \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. That a valuation v satisfies a clock constraint g, written $v \models g$, is defined in the obvious way. We write $\mathcal{C}(X)$ for the set of clock constraints over X.

Definition 1. *An* open timed automaton *(OTA) is a tuple*[1] $\mathcal{A} = (L^{\mathcal{A}}, l_0^{\mathcal{A}}, \Sigma^{\mathcal{A}}, X_p^{\mathcal{A}} \uplus X_o^{\mathcal{A}}, I^{\mathcal{A}}, E^{\mathcal{A}})$ *where:*

- $L^{\mathcal{A}}$ *is a finite set of* locations*, with* $l_0^{\mathcal{A}} \in L^{\mathcal{A}}$ *the* initial location*,*
- $\Sigma^{\mathcal{A}}$ *is a finite alphabet,*
- $X^{\mathcal{A}} = X_p^{\mathcal{A}} \uplus X_o^{\mathcal{A}}$ *is a finite set of clocks, partitionned into* proper clocks $X_p^{\mathcal{A}}$ *and* observed clocks $X_o^{\mathcal{A}}$*; only proper clocks may be reset along transitions.*
- $I^{\mathcal{A}}\colon L^{\mathcal{A}} \to \mathcal{C}(X^{\mathcal{A}})$ *assigns invariant constraints to locations.*
- $E^{\mathcal{A}} \subseteq L^{\mathcal{A}} \times \mathcal{C}(X^{\mathcal{A}}) \times \Sigma^{\mathcal{A}} \times 2^{X_p^{\mathcal{A}}} \times L^{\mathcal{A}}$ *is a finite set of* transitions*. For* $e = (l, g, a, X', l') \in E^{\mathcal{A}}$*, we write* $\mathsf{act}(e) = a$*.*

An Open Timed Automaton with Inputs and Outputs (OTAIO) is an OTA in which $\Sigma^{\mathcal{A}} = \Sigma_?^{\mathcal{A}} \uplus \Sigma_!^{\mathcal{A}} \uplus \Sigma_\tau^{\mathcal{A}}$ *is the disjoint union of* input actions *in* $\Sigma_?^{\mathcal{A}}$ *(noted* $?a$*,* $?b$*, ...),* output actions *in* $\Sigma_!^{\mathcal{A}}$ *(noted* $!a$*,* $!b$*, ...), and* internal actions *in* $\Sigma_\tau^{\mathcal{A}}$ *(noted* τ_1*,* τ_2*, ...) We write* $\Sigma_{\mathsf{obs}} = \Sigma_? \uplus \Sigma_!$ *for the alphabet of* observable *actions. Finally, a Timed Automaton (TA) (resp. a Timed Automaton with Inputs and Outputs (TAIO)) is an OTA (resp. an OTAIO) with no observed clocks.*

TAIOs will be sufficient to model most objects, but the ability of OTAIOs to observe other clocks will be essential for test purposes (see Section 3.1), which need to synchronize with the specification.

Let $\mathcal{A} = (L, l_0, \Sigma, X_p \uplus X_o, I, E)$ be an OTA. Its *semantics* is defined as an infinite-state transition system $\mathcal{T}^{\mathcal{A}} = (S^{\mathcal{A}}, s_0^{\mathcal{A}}, \Gamma^{\mathcal{A}}, \to^{\mathcal{A}})$ where:

- $S^{\mathcal{A}} = \{(l, v) \in L \times \mathbb{R}_{\geq 0}^X \mid v \models I(l)\}$ is the (infinite) set of *configurations*, with initial configuration $s_0^{\mathcal{A}} = (l_0, \overline{0}_X)$.
- $\Gamma^{\mathcal{A}} = \mathbb{R}_{\geq 0} \uplus (E \times 2^{X_o})$ is the set of *transitions labels.*
- $\to^{\mathcal{A}} \subseteq S^{\mathcal{A}} \times \Gamma^{\mathcal{A}} \times S^{\mathcal{A}}$ is the *transition relation.* It is defined as the union of

[1] For this and the following definitions, we may omit to mention superscripts when the corresponding automaton is clear from the context.

- the set of transitions corresponding to *time elapses*: it contains all triples $((l,v),\delta,(l',v')) \in S^{\mathcal{A}} \times \mathbb{R}_{\geq 0} \times S^{\mathcal{A}}$ for which $l = l'$ and $v' = v + \delta$. By definition of $S^{\mathcal{A}}$, both v and v' satisfy the invariant $I(l)$.
- the set of transitions corresponding to *discrete moves*: it contains all triples $((l,v),(e,X'_o),(l',v')) \in S^{\mathcal{A}} \times (E \times 2^{X_o}) \times S^{\mathcal{A}}$ such that, writing $e = (m,g,a,X'_p,m')$, it holds $m = l$, $m' = l'$, $v \models g$, and $v' = v_{[X'_p \cup X'_o \leftarrow 0]}$. Again, by definition, $v \models I(l)$ and $v' \models I(l')$.

An OTA has no control over its observed clocks, the intention being to synchronize them later in a product (see Def. 2). Hence, when a discrete transition is taken, any set X'_o of observed clocks may be reset. When dealing with plain TAs, where X_o is empty, we may write $(l,v) \xrightarrow{e} (l',v')$ in place of $(l,v) \xrightarrow{(e,\emptyset)} (l',v')$.

A *partial run* of $\mathcal{A}$ is a (finite or infinite) sequence of transitions in $\mathcal{T}^{\mathcal{A}}$ $\rho = ((s_i,\gamma_i,s_{i+1}))_{1\leq i<n}$, with $n \in \mathbb{N} \cup \{+\infty\}$. We write $\mathsf{first}(\rho)$ for s_1 and, when $n \in \mathbb{N}$, $\mathsf{last}(\rho)$ for s_n. A *run* is a partial run starting in the initial configuration $s_0^{\mathcal{A}}$. The duration of ρ is $\mathsf{dur}(\rho) = \sum_{\gamma_i \in \mathbb{R}_{\geq 0}} \gamma_i$. In the sequel, we only consider TAs in which any infinite run has infinite duration. We note $\mathsf{Ex}(\mathcal{A})$ for the set of runs of $\mathcal{A}$ and $\mathsf{pEx}(\mathcal{A})$ the subset of partial runs.

State s is *reachable* from state s' when there exists a partial run from s' to s. We write $\mathsf{Reach}(\mathcal{A},S')$ for the set of states that are reachable from some state in S', and $\mathsf{Reach}(\mathcal{A})$ for $\mathsf{Reach}(\mathcal{A},\{s_0^{\mathcal{A}}\})$.

The *(partial) sequence* associated with a (partial) run $\rho = ((s_i,\gamma_i,s'_i))_i$ is $\mathsf{seq}(\rho) = (\mathsf{proj}(\gamma_i))_i$, where $\mathsf{proj}(\gamma) = \gamma$ if $\gamma \in \mathbb{R}_{\geq 0}$, and $\mathsf{proj}(\gamma) = (a, X'_p \cup X'_o)$ if $\gamma = ((l,g,a,X'_p,l'),X'_o)$. We write $\mathsf{pSeq}(\mathcal{A}) = \mathsf{proj}(\mathsf{pEx}(\mathcal{A}))$ and $\mathsf{Seq}(\mathcal{A}) = \mathsf{proj}(\mathsf{Ex}(\mathcal{A}))$ for the sets of (partial) sequences of $\mathcal{A}$. We write $s \xrightarrow{\mu} s'$ when there exists a (partial) finite run ρ such that $\mu = \mathsf{proj}(\rho)$, $\mathsf{first}(\rho) = s$ and $\mathsf{last}(\rho) = s'$, and write $\mathsf{dur}(\mu)$ for $\mathsf{dur}(\rho)$. We write $s \xrightarrow{\mu}$ when $s \xrightarrow{\mu} s'$ for some s.

If $\mathcal{A}$ is a TAIO, the *trace* of a (partial) sequence corresponds to what can be observed by the environment, namely delays and observable actions. The trace of a sequence is the limit of the following inductive definition, for $\delta_i \in \mathbb{R}_{\geq 0}$, $a \in \Sigma_{obs}$, $\tau \in \Sigma_\tau$, $X' \subseteq X$, and a partial sequence μ:

$$\begin{aligned}
\mathsf{Trace}(\delta_1 ... \delta_k) &= \textstyle\sum_{i=1}^{k} \delta_i \qquad \text{(in particular } \mathsf{Trace}(\epsilon) = 0) \\
\mathsf{Trace}(\delta_1 ... \delta_k.(\tau,X').\mu) &= (\textstyle\sum_{i=1}^{k} \delta_i) \cdot \mathsf{Trace}(\mu) \\
\mathsf{Trace}(\delta_1 ... \delta_k.(a,X').\mu) &= (\textstyle\sum_{i=1}^{k} \delta_i) \cdot a \cdot \mathsf{Trace}(\mu)
\end{aligned}$$

We note $\mathsf{Traces}(\mathcal{A}) = \mathsf{Trace}(\mathsf{Seq}(\mathcal{A}))$ the set of traces corresponding to runs of $\mathcal{A}$ and $\mathsf{pTraces}(\mathcal{A})$ the subset of traces corresponding to partial runs. Two OTAIOs are said to be *trace-equivalent* if they have the same sets of traces. We furthermore define, for an OTAIO $\mathcal{A}$, a trace σ and a configuration s:

- $\mathcal{A}$ after $\sigma = \{s \in S \mid \exists \mu \in \mathsf{Seq}(\mathcal{A}), s_0 \xrightarrow{\mu} s \wedge \mathsf{Trace}(\mu) = \sigma\}$ is the set of all configurations that can be reached when σ has been observed from $s_0^{\mathcal{A}}$.
- $\mathsf{enab}(s) = \{e \in E^{\mathcal{A}} \mid s \xrightarrow{e}\}$ is the set of transitions enabled in s.

- $\mathsf{elapse}(s) = \{t \in \mathbb{R}_{\geq 0} \mid \exists \mu \in (\mathbb{R}_{\geq 0} \cup (\Sigma_\tau \times 2^X))^*, s \xrightarrow{\mu} \wedge \mathsf{dur}(\mu) = t\}$ is the set of delays that can be observed from location s without any observation.
- $\mathsf{out}(s) = \{a \in \Sigma_! \mid \exists e \in \mathsf{enab}(s), \mathsf{act}(e) = a\} \cup \mathsf{elapse}(s)$ is the set of possible outputs and delays that can be observed from s. For $S' \subseteq S$, we note $\mathsf{out}(S') = \bigcup_{s \in S'} \mathsf{out}(s)$.
- $\mathsf{in}(s) = \{a \in \Sigma_! \mid \exists e \in \mathsf{enab}(s), \mathsf{act}(e) = a\}$ is the set of possible inputs that can be proposed when arriving in s. For $S' \subseteq S$, we note $\mathsf{in}(S') = \bigcup_{s \in S'} \mathsf{in}(s)$.

We now define some useful sub-classes of OTAIOs. An OTAIO $\mathcal{A}$ is said

- *deterministic* if for all $\sigma \in Traces(\mathcal{A})$, $\mathcal{A}$ after σ is a singleton;
- *determinizable* if there exists a trace-equivalence deterministic OTAIO;
- *complete* if $S = L \times \mathbb{R}^X_{\geq 0}$ (i.e., all invariants are always true) and for any $s \in S$ and any $a \in \Sigma$, it holds $s \xrightarrow{a, X'}$ for some $X' \subseteq X$;
- *input-complete* if for any $s \in \mathsf{Reach}(\mathcal{A})$, $\mathsf{in}(s) = \Sigma_?$;
- *non-blocking* if for any $s \in \mathsf{Reach}(\mathcal{A})$ and any non-negative real t, there is a partial run ρ from s involving no input actions (i.e., $\mathsf{proj}(\rho)$ is a sequence over $\mathbb{R}_{\geq 0} \cup (\Sigma_! \cup \Sigma_\tau) \times 2^X$) and such that $\mathsf{dur}(\rho) = t$;
- *repeatedly observable* if for any $s \in \mathsf{Reach}(\mathcal{A})$, there exists a partial run ρ from s such that $\mathsf{Trace}(\rho) \notin \mathbb{R}_{\geq 0}$.

The product of two OTAIOs extends the classical product of TAs.

Definition 2. *Given two OTAIOs* $\mathcal{A} = (L^\mathcal{A}, l_0^\mathcal{A}, \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, X_p^\mathcal{A} \uplus X_o^\mathcal{A}, I^\mathcal{A}, E^\mathcal{A})$ *and* $\mathcal{B} = (L^\mathcal{B}, l_0^\mathcal{B}, \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, X_p^\mathcal{B} \uplus X_o^\mathcal{B}, I^\mathcal{B}, E^\mathcal{B})$ *over the same alphabets, their* product *is the OTAIO* $\mathcal{A} \times \mathcal{B} = (L^\mathcal{A} \times L^\mathcal{B}, (l_0^\mathcal{A}, l_0^\mathcal{B}), \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, (X_p^\mathcal{A} \cup X_p^\mathcal{B}) \uplus ((X_o^\mathcal{A} \cup X_o^\mathcal{B}) \setminus (X_p^\mathcal{A} \cup X_p^\mathcal{B})), I, E)$ *where* $I \colon (l_1, l_2) \mapsto I^\mathcal{A}(l_1) \wedge I^\mathcal{B}(l_2)$ *and* E *is the (smallest) set such that for each* $(l^1, g^1, a, X_p'^1, l'^1) \in E^\mathcal{A}$ *and* $(l^2, g^2, a, X_p'^2, l'^2) \in E^\mathcal{B}$, E *contains* $((l^1, l^2), g^1 \wedge g^2, a, X_p'^1 \cup X_p'^2, (l'^1, l'^2))$.

The product of two OTAIOs corresponds to the intersection of the sequences of the orginal OTAIOs, *i.e.* $\mathsf{Seq}(\mathcal{A} \times \mathcal{B}) = \mathsf{Seq}(\mathcal{A}) \cap \mathsf{Seq}(\mathcal{B})$ [BSJK15].

2.2 Timed games

We introduce timed game automata [AMPS98], which we later use to turn the test artifacts into games between the tester (controlling the environment) and the implementation, on an arena constructed from the specification.

Definition 3. *A* timed game automaton *(TGA) is a timed automaton* $\mathcal{G} = (L, l_0, \Sigma_c \uplus \Sigma_u, X, I, E)$ *where* $\Sigma = \Sigma_c \uplus \Sigma_u$ *is partitioned into actions that are controllable (*Σ_c*) and uncontrollable (*Σ_u*) by the player.*

All the notions of runs and sequences defined previously for TAs are extended to TGAs, with the interpretation of Σ_c as inputs and Σ_u as outputs.

Definition 4. *Let $\mathcal{G} = (L, l_0, \Sigma_c \uplus \Sigma_u, X, I, E)$ be a TGA. A* strategy *for the player is a partial function $f\colon Ex(\mathcal{G}) \to \mathbb{R}_{\geq 0} \times (\Sigma_c \cup \{\bot\}) \setminus \{(0, \bot)\}$ such that for any finite run ρ, letting $f(\rho) = (\delta, a)$, $\delta \in \mathsf{elapse}(\mathsf{last}(\rho))$ is a possible delay from $\mathsf{last}(\rho)$, and there is an a-transition available from the resulting configuration (unless $a = \bot$).*

Strategies give rise to sets of runs of $\mathcal{G}$, defined as follows:

Definition 5. *Let $\mathcal{G} = (L, l_0, \Sigma, X, I, E)$ be a TGA, f be a strategy over $\mathcal{G}$, and s be a configuration. The* set of outcomes *of f from s, noted $\mathsf{Outcome}(s, f)$, is the smallest subset of partial runs starting from s containing the empty partial run from s (whose last configuration is s), and s.t. for any $\rho \in \mathsf{Outcome}(s, f)$, letting $f(\rho) = (\delta, a)$ and $\mathsf{last}(\rho) = (l, v)$, we have*

- $\rho \cdot ((l, v), \delta, (l, v+\delta')) \cdot ((l, v+\delta'), e, (l', v')) \in \mathsf{Outcome}(s, f)$ *for any* $0 \leq \delta' \leq \delta$ *and* $\mathsf{act}(e) \in \Sigma_u$ *such that* $((l, v + \delta'), e, (l', v')) \in \mathsf{pEx}(\mathcal{A})$*;*
- *and*
  - *either $a = \bot$, and $\rho \cdot ((l, v), \delta, (l, v + \delta)) \in \mathsf{Outcome}(s, f)$;*
  - *or $a \in \Sigma_c$, and $\rho \cdot ((l, v), \delta, (l, v+\delta)) \cdot ((l, v+\delta), e, (l', v')) \in \mathsf{Outcome}(s, f)$ with $\mathsf{act}(e) = a$;*

An infinite partial run is in $\mathsf{Outcome}(s, f)$ if infinitely many of its finite prefixes are.

In this paper, we will be interested in reachability winning conditions (under particular conditions). In the classical setting, the set of winning configurations can be computed iteratively, starting from the target location and computing controllable predecessors in a backward manner. The computation can be performed on regions, so that it terminates (in exponential time) [AMPS98,CDF+05]. We extend this approach to our test-generation framework in Section 4.

3 Testing framework

We now present the testing framework, defining *(i)* the main testing artifacts *i.e.* specifications, implementations, test purposes, and test cases, along with the assumptions on them; *(ii)* a conformance relation relating implementations and specifications. The combination of the test purposes and the specification and the construction of an approximate deterministic tester is afterward explained.

3.1 Test context

We use TAIOs as models for specifications, implementations and test cases, and OTAIOs for test purposes. This allows to define liberal test purposes, and on a technical side, gives a unity to the manipulated objects.

In order to enforce the occurrence of conclusive verdicts, we equip specifications with *restart transitions*, corresponding to a system shutdown and restart, and assume that from any (reachable) configuration, a restart is always reachable.

Definition 6. *A* specification with restarts *(or simply* specification*) on* $(\Sigma_?, \Sigma_!, \Sigma_\tau)$ *is a non-blocking, repeatedly-observable TAIO* $\mathcal{S} = (L^{\mathcal{S}}, l_0^{\mathcal{S}}, (\Sigma_? \cup \{\zeta\}) \uplus \Sigma_! \uplus \Sigma_\tau, X_p^{\mathcal{S}}, I^{\mathcal{S}}, E^{\mathcal{S}})$ *where* $\zeta \notin \Sigma_?$ *is the restart action. We let* $\mathsf{Restart}^{\mathcal{S}} = E^{\mathcal{S}} \cap (L^{\mathcal{S}} \times G_{M^{\mathcal{S}}}(X^{\mathcal{S}}) \times \{\zeta\} \times \{X_p^{\mathcal{S}}\} \times \{l_0^{\mathcal{S}}\})$ *be the set of* ζ*-transitions, and it is assumed that from any reachable configuration, there exists a finite partial execution containing* ζ*, i.e. for any* $s \in \mathsf{Reach}(\mathcal{S})$*, there exists* μ *s.t.* $s \xrightarrow{\mu \cdot \zeta} s_0^{\mathcal{S}}$*.*

The non-blocking hypothesis rules out "faulty" specifications having no conformant physically-possible implementation. Repeated-observability will be useful for technical reasons, when analyzing the exhaustiveness property of test cases. Our assumption on ζ-transitions entails:

Proposition 7. *Let* $\mathcal{S}$ *be a specification with restarts. Then* $Reach(\mathcal{T}_{\mathcal{S}})$ *is strongly-connected.*

Example 1. Figure 1 is an example of specification for a conveyor belt. After a maximum time of 2 units (depending for example on their weight), packages reach a sorting point where they are automatically sorted between packages to reject and packages to ship. Packages to reject go to waste, while packages to ship are sent to a boarding platform, where an operator can send them to two different destinations. If the operator takes more than 3 units of time to select a destination, the package goes past the boarding platform and restarts the process.

In practice, test purposes are used to describe the intention of test cases, typically behaviours one wants because they must be correct and/or an error is suspected. In our formal testing framework, we describe them with OTAIOs that observe the specification together with accepting locations.

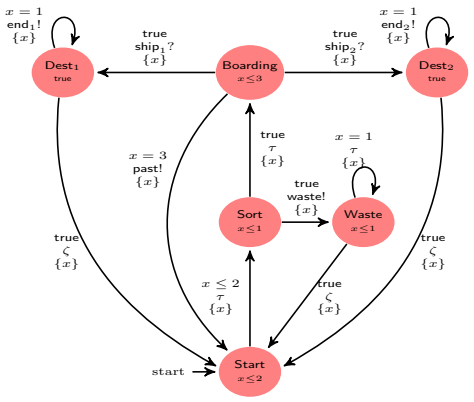

Fig. 1. A conveyor belt specification.

Definition 8. *Given a specification* $\mathcal{S} = (L^{\mathcal{S}}, l_0^{\mathcal{S}}, (\Sigma_? \cup \{\zeta\}) \uplus \Sigma_! \uplus \Sigma_\tau, X_p^{\mathcal{S}}, I^{\mathcal{S}}, E^{\mathcal{S}} \uplus \mathsf{Restart})$*, a* test purpose for $\mathcal{S}$ *is a pair* $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$ *where* $\mathcal{TP} = (L^{\mathcal{TP}}, l_0^{\mathcal{TP}}, \Sigma_? \cup \{\zeta\} \uplus \Sigma_! \uplus \Sigma_\tau, X_p^{\mathcal{TP}} \uplus X_p^{\mathcal{S}}, I^{\mathcal{TP}}, E^{\mathcal{TP}})$ *is a complete OTAIO together with a subset* $\mathsf{Accept}^{\mathcal{TP}} \subseteq L^{\mathcal{TP}}$ *of accepting locations, and such that transitions carrying restart actions* ζ *reset all proper clocks and return to the initial state (i.e., for any* ζ*-transition* $(l, g, \zeta, X', l') \in E$*, it must be* $X' = X_p^{\mathcal{TP}}$ *and* $l' = l_0^{\mathcal{TP}}$*).*

In the following, we may simply write $\mathcal{TP}$ in place of $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$. We force test purposes to be complete because they should never constrain the runs of the specification they observe, but should only label the accepted behaviours to be tested. Test purposes observe exactly the clocks of the specification in order to synchronize with them, but cannot reset them.

Example 2. Figure 2 is a test purpose for our conveyor-belt example. We want to be sure that it is possible to ship a package to destination 2 in less than 5 time units, while avoiding to go in waste. The Accept set is limited to a location, named Accept. We note oth the set of transitions that reset no clocks, and is enabled for an action other than ζ when no other transition is possible for this action. This set serves to complete the test purpose. The test purpose has a proper clock y.

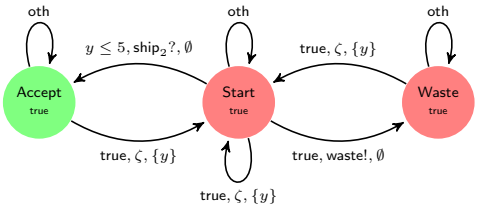

Fig. 2. A test purpose for the conveyor belt.

In practice, conformance testing links a mathematical model, the specification, and a black-box implementation, that is a real-life *physical* object observed by its interactions with the environment. In order to formally reason about conformance, one needs to bridge the gap between the mathematical world and the physical world. We then assume that the implementation corresponds to an unknown TAIO.

Definition 9. *Let* $\mathcal{S} = (L^{\mathcal{S}}, l_0^{\mathcal{S}}, (\Sigma_? \cup \{\zeta\}) \uplus \Sigma_! \uplus \Sigma_\tau, X_p^{\mathcal{S}}, I^{\mathcal{S}}, E^{\mathcal{S}} \cup \mathsf{Restart})$ *be a specification TAIO. An* implementation *of* $\mathcal{S}$ *is an input-complete and non-blocking TAIO* $\mathcal{I} = (L^{\mathcal{I}}, l_0^{\mathcal{I}}, (\Sigma_? \cup \{\zeta\}) \uplus \Sigma_! \uplus \Sigma_\tau^{\mathcal{I}}, X_p^{\mathcal{I}}, I^{\mathcal{I}}, E^{\mathcal{I}})$*. We note* $\mathcal{I}(\mathcal{S})$ *the set of possible implementations of* $\mathcal{S}$*.*

The hypotheses made on implementations are not restrictions, but model real-world contingencies: the environment might always provide any input and the system cannot alter the course of time.

Having defined the necessary objects, it is now possible to introduce the *timed input-output conformance* (tioco) relation [KT09]. Intuitively, it can be understood as "after any specified behaviour, outputs and delays of the implementation should be specified".

Definition 10. *Let* $\mathcal{S}$ *be a specification and* $\mathcal{I} \in \mathcal{I}(\mathcal{S})$*. We say that* $\mathcal{I}$ conforms to $\mathcal{S}$ *for tioco, and write* $\mathcal{I}$ *tioco* $\mathcal{S}$ *when:*

$$\forall \sigma \in \mathit{Traces}(\mathcal{S}), \mathit{out}(\mathcal{I} \text{ after } \sigma) \subseteq \mathit{out}(\mathcal{S} \text{ after } \sigma)$$

Note that it is not assumed that restarts are well implemented: if they are not, it is significant only if it induces non-conformant behaviours.

3.2 Combining specifications and test purposes

Now that the main objects are defined, we explain how the behaviours targeted by the test purpose $\mathcal{TP}$ are characterized on the specification $\mathcal{S}$ by the construction of the product OTAIO $\mathcal{P} = \mathcal{S} \times \mathcal{TP}$. Since $\mathcal{S}$ is a TAIO and the observed clocks of $\mathcal{TP}$ are exactly the clocks of $\mathcal{S}$, the product $\mathcal{P}$ is actually a TAIO. Furthermore, since $\mathcal{TP}$ is complete, $\mathsf{Seq}(\mathcal{P}) = \mathsf{Seq}(\mathcal{S})$. This entails that $\mathcal{I}$ tioco $\mathcal{S}$ is equivalent to $\mathcal{I}$ tioco $\mathcal{P}$. Note in particular that ζ of $\mathcal{S}$ synchronize with ζ of $\mathcal{TP}$, which are available everywhere.

By defining accepting locations in the product by $\mathsf{Accept}^{\mathcal{P}} = L^{\mathcal{S}} \times \mathsf{Accept}^{\mathcal{TP}}$, we get that sequences accepted in $\mathcal{P}$ are exactly sequences of $\mathcal{S}$ accepted by $\mathcal{TP}$.

Example 3. Fig. 3 represents the product of the conveyor-belt specification of Fig. 1 and the test purpose of Fig. 2. All nodes are named by the first letters of the corresponding states of the specification (first) and of the test purpose. The only accepting location is (D_2, A).

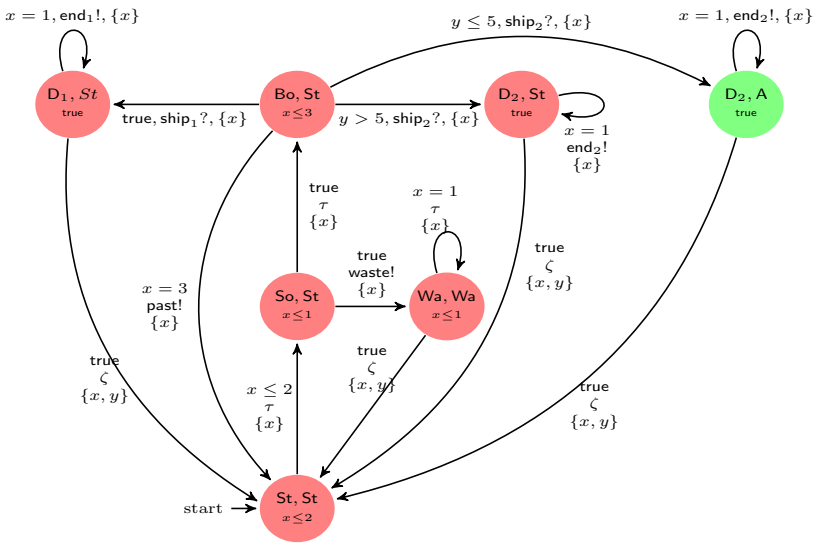

Fig. 3. Product of the conveyor belt specification and the presented test purpose.

We make one final hypothesis: we consider only pairs of specifications $\mathcal{S}$ and test purposes $\mathcal{TP}$ whose product $\mathcal{P}$ can be exactly determinized by the determinization game presented in [BSJK15]. This restriction is necessary for technical reasons: if the determinization is approximated, we cannot ensure that restarts are still reachable in general. Notice that it is satisfied in several classes of automata, such as strongly non-zeno automata, integer-reset automata, or event-recording automata.

Given the product $\mathcal{P} = \mathcal{S} \times \mathcal{TP}$, let $\mathcal{DP}$ be its exact determinization. In this case, $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$, hence the reachability of ζ transitions is preserved. Moreover the traces leading to $\mathsf{Accept}^{\mathcal{DP}}$ and $\mathsf{Accept}^{\mathcal{P}}$ are the same.

Example 4. The automaton in Fig. 4 is a deterministic approximation of the product presented in Fig. 3. The internal transitions have collapsed, leading to an augmented Start locality.

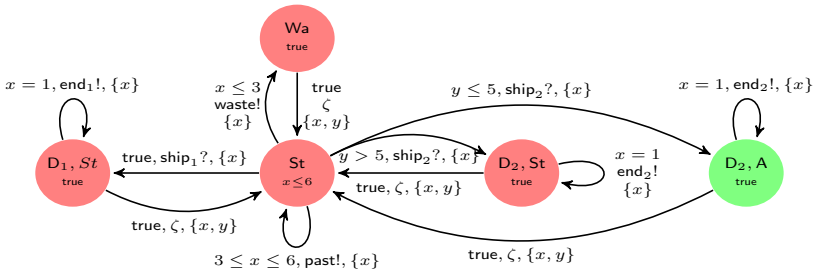

Fig. 4. A deterministic approximation of the product.

3.3 Accounting for failure

At this stage of the process, we dispose of a deterministic and fully-observable TAIO $\mathcal{DP}$ having exactly the same traces as the original specification, and having a subset of its localities labelled as accepting for the test purpose. From this TAIO, we aim to build a tester, that can be able to monitor the implementation, feeding it with inputs and selecting verdicts from the returned outputs.

$\mathcal{DP}$ models the accepted traces with $\mathsf{Accept}^{\mathcal{DP}}$. In order to also explicitely model faulty behaviours (unspecified outputs after a specified trace), we now complete $\mathcal{DP}$ with respect to its output alphabet, by adding an explicit Fail location. We call this completed TAIO the *objective-centered tester*.

Definition 11. *Given a deterministic TAIO* $\mathcal{DP} = (L^{\mathcal{DP}}, l_0^{\mathcal{DP}}, \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, X_p^{\mathcal{DP}}, I^{\mathcal{DP}}, E^{\mathcal{DP}})$*, we construct its objective-centered tester* $\mathcal{OT} = (L^{\mathcal{DP}} \cup \{\mathsf{Fail}\}, l_0^{\mathcal{DP}}, \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, X_p^{\mathcal{DP}}, I^{\mathcal{OT}}, E^{\mathcal{OT}})$ *where* $I^{\mathcal{OT}}(l) = \mathsf{true}$*. The set of transitions* $E^{\mathcal{OT}}$ *is defined from* $E^{\mathcal{DP}}$ *by:*

$$E^{\mathcal{OT}} = E^{\mathcal{DP}} \cup \bigcup_{\substack{l \in L^{\mathcal{DP}} \\ a \in \Sigma_!^{\mathcal{DP}}}} \{(l, g, a, \emptyset, \mathsf{Fail}) \mid g \in \overline{G}_{a,l}\} \cup \{(\mathsf{Fail}, \mathsf{true}, a, \emptyset, \mathsf{Fail}) \mid a \in \Sigma^{\mathcal{DP}}\}$$

where for each a *and* l*,* $\overline{G}_{a,l}$ *is a set of guards complementing the set of all valuations* v *for which an* a*-transition is available from* (l, v) *(notice that* $\overline{G}_{a,l}$ *generally is non-convex, so that it cannot be represented by a single guard).*

Verdicts are defined on the configurations of $\mathcal{OT}$ *as follows:*

– $\boldsymbol{Pass} = \bigcup_{l \in \mathit{Accept}^{\mathcal{DP}}}(\{l\} \times I^{\mathcal{DP}}(l))$,
– $\boldsymbol{Fail} = \{\mathit{Fail}\} \times \mathbb{R}_{\geq 0} \cup \bigcup_{l \in L^{\mathcal{DP}}} \left(\{l\} \times \left(\mathbb{R}_{\geq 0}^{X_p} \setminus I^{\mathcal{DP}}(l)\right)\right)$.

Notice that we do not define the usual ***Inconclusive*** verdicts (*i.e.* configurations in which we cannot conclude to non-conformance, nor accept the run with respect to the test purpose) as we will enforce the apparition of ***Pass*** or ***Fail***. ***Pass*** corresponds to behaviours accepted by the test purpose, while ***Fail*** corresponds to non-conformant behaviours. Note that $\mathcal{OT}$ inherits the interesting structural properties of $\mathcal{DP}$. More importantly, ζ is always reachable as long as no verdict has been emited, and $\mathcal{OT}$ is repeatedly-observable out of ***Fail***.

It remains to say that $\mathcal{OT}$ and $\mathcal{DP}$ model the same behaviours. Obviously, their sets of traces differ, but the traces added in $\mathcal{OT}$ precisely correspond to runs reaching ***Fail***. We now define a specific subset of runs, sequences and traces corresponding to traces that are meant to be accepted by the specification.

Definition 12. *A run ρ of an objective-centered tester $\mathcal{OT}$ is said* conformant *if it does not reach* ***Fail****. We note $\mathit{Ex}_{\mathit{conf}}(\mathcal{OT})$ the set of conformant runs of $\mathcal{OT}$, and $\mathit{Seq}_{\mathit{conf}}(\mathcal{OT})$ (resp. $\mathit{Traces}_{\mathit{conf}}(\mathcal{OT})$) the corresponding sequences (resp. traces). We note $\mathit{Ex}_{\mathit{fail}}(\mathcal{OT}) = \mathit{Ex}(\mathcal{OT}) \setminus \mathit{Ex}_{\mathit{conf}}(\mathcal{OT})$ and similarly for the sequences and traces.*

The conformant traces are exactly those specified by $\mathcal{DP}$, i.e. $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ and correspond to executions tioco-conformant with the specification, while $\mathsf{Ex}_{\mathsf{fail}}$ are runs where a non-conformance is detected.

4 Translating objectives into games

In this section, we interpret objective-centered tester into games between the tester and the implementation and propose strategies that try to avoid control losses. We then introduce a scope in which the tester always has a winning strategy, and discuss the properties of the resulting test cases (*i.e.* game structure and built strategy).

We want to enforce conclusive verdicts when running test cases, i.e. either the implementation does not conform to its specification (***Fail*** verdict) or the awaited behaviour appears (***Pass*** verdict). We thus say that an execution ρ is winning for the tester if it reaches a ***Fail*** or ***Pass*** configuration and note $\mathsf{Win}(\mathcal{G})$ the set of such executions. In the following, we consider the TGA $\mathcal{G}^{\mathcal{OT}} = (L^{\mathcal{OT}}, l_0^{\mathcal{OT}}, \Sigma_?^{\mathcal{OT}} \uplus \Sigma_!^{\mathcal{OT}}, X_p, I^{\mathcal{OT}}, E^{\mathcal{OT}})$ where the controllable actions are the inputs $\Sigma_c = \Sigma_?^{\mathcal{OT}}$ and the uncontrollable actions are the outputs $\Sigma_u = \Sigma_!^{\mathcal{OT}}$.

4.1 Rank-lowering strategy

In this part, we restrict our discussion to TGAs where ***Pass*** configurations are reachable (when seen as plain TAs). Indeed, if none can be reached, and we will discuss the fact that the proposed method can detect this fact, trying to construct

a strategy seeking a ***Pass*** verdict is hopeless. This is a natural restriction, as it only rules out unsatisfiable test purposes.

The tester cannot force the occurrence of a non-conformance (as he does not control outputs and delays), and hence cannot push the system into a ***Fail*** configuration. A strategy for the tester should thus target the ***Pass*** set in a partially controlable way, while monitoring ***Fail***. For that purpose, we define a hierarchy of configurations, depending on their "distance" to ***Pass***. This uses a backward algorithm, for which we define the predecessors of a configuration.

Given a set of configurations $S' \subseteq S$ of $\mathcal{G}^{\mathcal{OT}}$, we define three kinds of predecessors, letting $\overline{V}$ denote the complement of V:

- discrete predecessors by a sub-alphabet $\Sigma' \subseteq \Sigma$:

  $$\mathsf{Pred}_{\Sigma'}(S') = \{(l,v) \mid \exists a \in \Sigma',\ \exists (l,a,g,X',l') \in E, v \models g \wedge (l', v_{[X' \leftarrow 0]}) \in S'\}$$

- timed predecessors, while avoiding a set V of configurations:

  $$\mathsf{tPred}(S', V) = \{(l,v) \mid \exists \delta \in \mathbb{R}_{\geq 0},\ (l, v+\delta) \in S' \wedge \forall\, 0 \leq \delta' \leq \delta.\ (l, v+\delta') \notin V\}$$

  We furthermore note $\mathsf{tPred}(S') = \mathsf{tPred}(S', \emptyset)$.
- final timed predecessors are defined for convenience (see below):

  $$\mathsf{ftPred}(S') = \mathsf{tPred}(\textbf{\textit{Fail}}, \mathsf{Pred}_{\Sigma_u}(\overline{S'})) \cup \overline{\mathsf{tPred}(\mathsf{Pred}_{\Sigma}(\overline{S'}))}$$

The final timed predecessors correspond to situations where the system is 'cornered', having the choice between taking an uncontrollable transition to S' (as no uncontrollable transition to $\overline{S'}$ will be available) or reach ***Fail***. Such situations are not considered as control losses, as the system can only take a beneficial transition for the tester (either by going to S' or to ***Fail***). Note that tPred and ftPred need not return convex sets, but are efficiently computable using Pred and simple set constructions [CDF+05]. Now, using these notions of predecessors, a hierarchy of configurations based on the 'distance' to ***Pass*** is defined.

Definition 13. *The sequence $(W_i^j)_{j,i}$ of sets of configurations is defined as:*

- $W_0^0 = \textbf{\textit{Pass}}$
- $W_{i+1}^j = \pi(W_i^j)$ *with* $\pi(S') = \mathit{tPred}\left(S' \cup \mathit{Pred}_{\Sigma_c}(S'), \mathit{Pred}_{\Sigma_u}(\overline{S'})\right) \cup \mathit{ftPred}(S')$
- $W_0^{j+1} = \mathit{tPred}(W_\infty^j \cup \mathit{Pred}_\Sigma(W_\infty^j))$ *with* W_∞^j *the limit*[2] *of the sequence* $(W_i^j)_i$.

In this hierarchy, j corresponds to the minimal number of *control losses* the tester has to go through (in the worst case) in order to reach ***Pass***, and i corresponds to the minimal number of steps before the next control loss (or to ***Pass***). The W_0^{j+1} are considered 'control losses' as the implementation might take an output transition leading to an undesirable configuration (higher on the hierarchy). On the other hand, in the construction of W_i^j the tester keep a full control, as it is not possible to reach such bad configuration with an incontrolable transition. Notice that the sequence (W_i^j) is an increasing sequence of regions, and hence can be computed in time exponential in $X^{\mathcal{OT}}$ and linear in $L^{\mathcal{OT}}$.

We then have the following property:

[2] The sequence $(W_i^j)_i$ is non-decreasing, and can be computed in terms of clock regions; hence the limit exists and is reached in a finite number of iterations [CDF+05].

Proposition 14. *There exists $i, j \in \mathbb{N}$ such that $\mathsf{Reach}(\mathcal{G}^{\mathcal{OT}}) \setminus \textbf{\textit{Fail}} \subseteq W_i^j$.*

As explained above, this property is based on the assumption that the ***Pass*** verdict is reachable. Nevertheless, if it is not it will be detected during the hierarchy construction that will converge to a fixpoint not including $s_0^{\mathcal{G}}$. As all the configurations in which we want to define a strategy are covered by the hierarchy, we can use it to define a partial order.

Definition 15. *Let $s \in \mathsf{Reach}(\mathcal{G}^{\mathcal{OT}}) \setminus \textbf{\textit{Fail}}$. The* rank *of s is:*

$$r(s) = (j_s = \arg\min_{j \in \mathbb{N}}(s \in W_\infty^j), i_s = \arg\min_{i \in \mathbb{N}}(s \in W_i^{j_s}))$$

For $r(s) = (j, i)$; j is the minimal number of control losses before reaching an accepting state, and i is the minimal number of steps in the strategy before the next control loss. We note $s \sqsubseteq s'$ when $r(s) \leq_{\mathbb{N}^2} r(s')$, where $\leq_{\mathbb{N}^2}$ is the lexical order on $\mathbb{N}^2$.

Proposition 16. *$\sqsubseteq$ is a partial order on $\mathsf{Reach}(\mathcal{G}^{\mathcal{OT}}) \setminus \textbf{\textit{Fail}}$.*

We dispose of a partial order on configurations, with ***Pass*** being the minimal elements. We use it to define a strategy trying to decrease the rank during the execution. For any $s \in S$, we write $r^-(s)$ for the largest rank such that $r^-(s) <_{\mathbb{N}^2} r(s)$, and $W^-(s)$ for the associated set in $(W_i^j)_{j,i}$. We (partially) order pairs $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$ according to δ.

Definition 17. *A strategy f for the tester is* rank-lowering *if, for any finite run ρ with $\mathsf{last}(\rho) = s = (l, v)$, it selects the lesser delay satisfying one of the following constraints:*

- *if $s \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_c}(W^-(s)))$,then $f(\rho) = (\delta, a)$ with $a \in \Sigma_c$ s.t. there exists $e \in E$ with $\mathsf{act}(e) = a$ and $s \xrightarrow{\delta} \xrightarrow{e} t$ with $t \in W^-(s)$, and δ is minimal in the following sense: if $s \xrightarrow{\delta'} \xrightarrow{e'} t'$ with $t' \in W^-(s)$ and $\delta' \leq \delta$, then $v + \delta$ and $v + \delta'$ belong to the same region;*
- *if $s \in \mathsf{tPred}(W^-(s))$,then $f(\rho) = (\delta, \bot)$ such that $s \xrightarrow{\delta} t$ with $t \in W^-(s)$, and δ is minimal in the same sense as above;*
- *otherwise $f(\rho) = (\delta, \bot)$ where δ is maximal in the same sense as above (maximal delay-successor region).*

The two first cases follow the construction of the W_i^j and propose the shortest behaviour leading to W^-. The third case corresponds, either to a configuration of ***Pass***, where W^- is undefined, or to a ftPred. Notice that (possibly several) rank-lowering strategies always exist.

Example 5. An example of a rank-lowering strategy on the automaton of Fig. 4 is: in $(\mathsf{D}_2, \mathsf{A})$, play $\bot$ (as W_0^0 has been reached); in St, play $(0, \mathsf{ship}_2?)$; in any other state, play $(0, \zeta)$. Note that Fig. 4 has not been completed in a objective-centered tester. This does not impact the strategies, as the transition to the failstates lead to a victory, but are not targeted by the strategies.

It is worth noting that even in a more general setup where the models are not equipped with ζ-transitions, as in [BSJK15], rank-lowering strategies may still be useful: as they are defined on the co-reachable set of Accept, they can still constitute test cases, and the configurations where they are not defined are exactly the configurations corresponding to a ***Fail*** verdict or to an ***Inconclusive*** verdict, i.e., no conclusions can be made since an accepting configuration cannot be reached.

4.2 Making rank-lowering strategies win

A rank-lowering strategy is generally not a winning strategy: it relies on the implementation fairly exploring its different possibilities and not repeatedly avoiding an enabled transition. In this section, fair runs are introduced, and the rank-lowering strategies are shown to be winning in this subset of the runs.

Lemma 18. *If $\mathcal{OT}$ is repeatedly-observable, then for all $\rho = ((s_i, \gamma_i, s_{i+1}))_{i\in\mathbb{N}} \in \mathit{Ex}(\mathcal{G})$ ending with an infinite sequence of delays, we have*[3]

$$\rho \in \mathit{Ex}_{\mathit{fail}}(\mathcal{G}) \vee \exists e \in E^{\mathcal{G}},\ \overset{\infty}{\exists}\ i \in \mathbb{N}, e \in \mathit{enab}(s_i).$$

This lemma ensures that we cannot end in a situation where no transitions can be taken, forcing the system to delay indefinitely. It will be used with the support of fairness. In order to introduce our notion of fairness, we define the infinite support of a run.

Definition 19. *Let ρ be an infinite run, its* infinite support $\mathit{Inf}(\rho)$ *is the set of regions appearing infinitely often in ρ.*

$$\mathit{Inf}((s_i, \gamma_i, s_{i+1})_{i\in\mathbb{N}}) = \{r \mid \overset{\infty}{\exists}\ i \in \mathbb{N},\ s_i \in r \vee (\gamma_i \in \mathbb{R}_{\geq 0} \wedge \exists s_i' \in r,\ \exists \delta_i < \gamma_i,\ s_i \xrightarrow{\delta_i} s_i')\}$$

The notion of enabled transitions and delay transitions are extended to regions as follows: for a region r, we let $\mathsf{enab}(r) = \mathsf{enab}(s)$ for any s in r, and write $r \xrightarrow{\mathtt{t}} r'$ for all time-successor region r' of r.

Definition 20. *An infinite run ρ in a TGA $\mathcal{G} = (L, l_0, \Sigma_u \uplus \Sigma_c, X, I, E)$ (with timed transitions system $\mathcal{T} = (S, s_0, \Gamma, \rightarrow_{\mathcal{T}})$) is said to be* fair *when:*

$$\forall e \in E, (\mathit{act}(e) \in \Sigma_u \Rightarrow (\exists r \in \mathit{Inf}(\rho), r \xrightarrow{e} r' \Rightarrow r' \in \mathit{Inf}(\rho))) \ \wedge$$
$$\forall r \in \mathit{Inf}(\rho), \exists \gamma \in (\mathit{enab}(r) \cap \{e \mid \mathit{act}(e) \in \Sigma_c\}) \cup \{\mathtt{t}\}), r \xrightarrow{\gamma} r' \wedge r' \in \mathit{Inf}(\rho)$$

We note $\mathit{Fair}(\mathcal{G})$ the set of fair runs of $\mathcal{G}$.

[3] In this expression, $\overset{\infty}{\exists}\ i \in \mathbb{N},\ \phi(i)$ means that $\phi(i)$ is true for infinitely many integers.

Fair runs model restrictions on the system runs corresponding to strategies of the system. The first part of the definition assures that any infinitely-enabled action of the implementation will be taken infinitely many times, while the second part ensures that the implementation will infinitely often let the tester play, by ensuring that a delay or controlable action will be performed. It matches the "strong fairness" notion used in model checking. Restricting to fair runs is sufficient to ensure a winning execution when the tester uses a rank-lowering strategy. Intuitively, combined with Lemma 18 and the repeated-observability assumption, it assures that the system will keep providing outputs until a verdict is reached, and allows to show the following property.

Proposition 21. *Rank-lowering strategies are winning on* $\mathsf{Fair}(\mathcal{G})$ *(i.e., all fair outcomes are winning).*

Under the hypothesis of a fair implementation, we thus have identified a test-case generation method, starting from the specification with restarts and the test purpose, and constructing a test case as a winning strategy on the game created from the objective-centered tester. The complexity of this method is exponential in the size of $\mathcal{DP}$. More precisely:

Proposition 22. *Given a deterministic product* $\mathcal{DP}$*,* $\mathcal{OT}$ *can be linearly computed from* $\mathcal{DP}$*, and the construction of a strategy relies on the construction of the* W_i^j *and is hence exponential in* $X^{\mathcal{DP}}$ *and linear in* $L^{\mathcal{DP}}$*.*

Note that if $\mathcal{DP}$ is obtained from $\mathcal{P}$ by the game presented in [BSJK15], then $L^{\mathcal{DP}}$ is doubly exponential in the size of $X^{\mathcal{S}} \uplus X^{\mathcal{TP}} \uplus X^{\mathcal{DP}}$ (notice that in the setting of [BSJK15], $X^{\mathcal{DP}}$ is a parameter of the algorithm).

4.3 Properties of the test cases

Having constructed strategies for the tester, and identified a scope of implementation behaviours that allows these strategies to enforce a conclusive verdict, we now study the properties obtained by the test generation method presented above. We call *test case* a pair $(\mathcal{G}, f)$ where $\mathcal{G}$ is the game corresponding to the objective-centered tester $\mathcal{OT}$, and f is a rank-lowering strategy on $\mathcal{G}$. We note $\mathcal{TC}(\mathcal{S}, \mathcal{TP})$ the set of possibles test cases generated from a specification $\mathcal{S}$ and a test purpose $\mathcal{TP}$, and $\mathcal{TC}(\mathcal{S})$ the test cases for any test purpose. Recall that it is assumed that the test purposes associated with a specification are restricted to those leading to a determnizable product. *Behaviours* are defined as the possible outcomes of a test case combined with an implementation, and model their parallel composition.

Definition 23. *Given a test case* $(\mathcal{G}, f)$ *and an implementation* $\mathcal{I}$*, their* behaviours *are the runs* $((s_i, s_i'), (e_i, e_i'), (s_{i+1}, s_{i+1}'))_i$ *such that* $((s_i, e_i, s_{i+1}))_i$ *is an outcome of* $(\mathcal{G}, f)$*,* $((s_i', e_i', s_{i+1}'))_i$ *is a run of* $\mathcal{I}$*, and for all* i*, either* $e_i = e_i'$ *if* $e_i \in \mathbb{R}_{\geq 0}$ *or* $\mathsf{act}(e_i) = \mathsf{act}(e_i')$ *otherwise. We write* $\mathsf{Behaviour}(\mathcal{G}, f, \mathcal{I})$ *for the set of behaviours of the test case* $(\mathcal{G}, f)$ *and of the implementation* $\mathcal{I}$*.*

We say that an implementation $\mathcal{I}$ fails a test case $(\mathcal{G}, f)$, and note $\mathcal{I}$ fails $(\mathcal{G}, f)$, when there exists a run in $\mathsf{Behaviour}(\mathcal{G}, f, \mathcal{I})$ that reaches ***Fail***. Our method is sound, that is, a conformant implementation cannot be detected as faulty.

Proposition 24. *The test-case generation method is* sound*: for any specification $\mathcal{S}$, it holds*

$$\forall \mathcal{I} \in \mathcal{I}(\mathcal{S}),\ \forall (\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S}),\ (\mathcal{I}\ \mathit{fails}\ (\mathcal{G}, f) \Rightarrow \neg(\mathcal{I}\ \mathit{tioco}\ \mathcal{S})).$$

The proofs of this property and the following one are based on the exact correspondance between ***Fail*** and the faulty behaviours of $\mathcal{S}$, and use the trace equivalence of the different models ($\mathcal{DP}$, $\mathcal{P}$ and $\mathcal{S}$) to conclude. As they exploit mainly the game structure, fairness is not used.

We define the notion of outputs after a trace in a behaviour, allowing to extend tioco to these objects and to state a strictness property. Intuitively, when a non-conformance appears it should be detected.

Definition 25. *Given a test case $(\mathcal{G}, f)$ and an implementation $\mathcal{I}$, for a trace σ:*

$$\mathit{out}(\mathit{Behaviour}(\mathcal{G}, f, \mathcal{I})\ \mathit{after}\ \sigma) = \{a \in \Sigma_! \cup \mathbb{R}_{\geq 0} \mid \exists \rho \in \mathit{Behaviour}(\mathcal{G}, f, \mathcal{I}),\ \mathit{Trace}(\rho) = \sigma \cdot a\}$$

Proposition 26. *The test generation method is* strict*: given a specification $\mathcal{S}$,*

$$\forall \mathcal{I} \in \mathcal{I}(\mathcal{S}),\ \forall (\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S}),\ \neg(\mathit{Behaviour}(\mathcal{G}, f, \mathcal{I})\ \mathit{tioco}\ \mathcal{S}) \Rightarrow \mathcal{I}\ \mathit{fails}\ (\mathcal{G}, f)$$

This method also enjoys a precision property: traces leading the test case to ***Pass*** are exactly traces conforming to the specification and accepted by the test purpose. The proof of this property uses the exact encoding of the Accept states and the definition of ***Pass***. As the previous two, it then propagates the property through the different test artifacts.

Proposition 27. *The test case generation method is* precise*: for any specification $\mathcal{S}$ and test purpose $\mathcal{TP}$ it can be stated that*

$$\forall (\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S}, \mathcal{TP}), \forall \sigma \in \mathit{Traces}(\mathit{Outcome}(s_0^{\mathcal{G}}, f)),$$
$$\mathcal{G}\ \mathit{after}\ \sigma \in \textbf{\textit{Pass}} \Leftrightarrow (\sigma \in \mathit{Traces}(\mathcal{S}) \wedge \mathcal{TP}\ \mathit{after}\ \sigma \cap \mathit{Accept}^{\mathcal{TP}} \neq \emptyset)$$

Lastly, this method is exhaustive in the sense that for any non-conformance, there exist a test case that allows to detect it, under fairness assumption.

Proposition 28. *The test generation method is* exhaustive*: for any exactly determinizable specification $\mathcal{S}$ and any implementation $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ making fair runs*

$$\neg(\mathcal{I}\ \mathit{tioco}\ \mathcal{S}) \Rightarrow \exists (\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S}), \mathcal{I}\ \mathit{fails}\ (\mathcal{G}, f).$$

To demonstrate this property, a test purpose is tailored to detect a given non-conformance, by targeting a related conformant trace.

5 Conclusion

This paper proposes a game approach to the controllability problem for conformance testing from timed automata (TA) specifications. It defines a test synthesis method that produces test cases whose aim is to maximize their control upon the implementation under test, while detecting non-conformance. Test cases are defined as strategies of a game between the tester and the implementation, based on the distance to the satisfaction of a test purpose, both in terms of number of transitions and potential control losses. Fairness assumptions are used to make those strategies winning and are proved sufficient to obtain the exhaustiveness of the test synthesis method, together with soundness, strictness and precision.

This paper opens numerous directions for future work. First, we intend to tackle partial observation in a more complete and practical way. One direction consists in finding weaker conditions under which approximate determinization [BSJK15] preserves strong connectivity, a condition for the existence of winning strategies. One could also consider a mixture of our model and the model of [DLL$^+$10] whose observer predicates are clearly adequate in some contexts. Quantitative aspects could also better meet practical needs. The distance to the goal could also include the time distance or costs of transitions, in particular to avoid restarts when they induce heavy costs but longer and cheaper paths are possible. The fairness assumption could also be refined. For now it is assumed on both the specification and the implementation. If the implementation does not implement some outputs, a tester could detect it with a bounded fairness assumption [Ram98], adapted to the timed context (after sufficiently many experiments traversing some region all outputs have been observed), thus allowing a stronger conformance relation with egality of output sets. A natural extension could also be to complete the approach in a stochastic view. Finally, we plan to implement the results of this work in an open tool for the analysis of timed automata, experiment on real examples and check the scalability of the method.

References

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proceedings of the 5th IFAC Cconference on System Structure and Control (SSC'98)*, pages 469–474. Elsevier, July 1998.

[BB04] Laura Brandán Briones and Ed Brinksma. A test generation framework for quiescent real-time systems. In Jens Grabowski and Brian Nielsen, editors, *Proceedings of the 4th International Workshop on Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *Lecture Notes in Computer Science*, pages 64–78. Springer-Verlag, September 2004.

[BJSK12] Nathalie Bertrand, Thierry Jéron, Amélie Stainer, and Moez Krichen. Offline test selection with test purposes for non-deterministic timed automata. *Logical Methods in Computer Science*, 8(4), 2012.

[BSJK15] Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen. A game approach to determinize timed automata. *Formal Methods in System Design*, 46(1):42–80, February 2015.

[CDF+05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer-Verlag, August 2005.

[CKL98] Richard Castanet, Ousmane Koné, and Patrice Laurençot. On-the-fly test generation for real time protocols. In *Proceedings of the International Conference On Computer Communications and Networks (ICCCN'98)*, pages 378–387. IEEE Comp. Soc. Press, October 1998.

[COG98] Rachel Cardell-Oliver and Tim Glover. A practical and complete algorithm for testing real-time systems. In Anders P. Ravn and Hans Rischel, editors, *Proceedings of the 5th Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, volume 1486 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, September 1998.

[DLL+10] Alexandre David, Kim Guldstrand Larsen, Shuhao Li, Marius Mikučionis, and Brian Nielsen. Testing real-time systems under uncertainty. In *Revised Papers of the 13th International Conference on Formal Methods for Components and Objects (FMCO'10)*, volume 6957 of *Lecture Notes in Computer Science*, pages 352–371. Springer-Verlag, December 2010.

[DLLN08a] Alexandre David, Kim G. Larsen, Shuhao Li, and Brian Nielsen. Cooperative testing of timed systems. In *Proceedings of the 4th Workshop on Model Based Testing (MBT'08)*, volume 220, pages 79–92, 2008.

[DLLN08b] Alexandre David, Kim G. Larsen, Shuhao Li, and Brian Nielsen. A game-theoretic approach to real-time system testing. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'08)*, pages 486–491, March 2008.

[ENDK02] Abdeslam En-Nouaary, Radhida Dssouli, and Ferhat Khendek. Timed wp-method: Testing real-time systems. *IEEE Transactions on Software Engineering*, 28(11):1023–1038, November 2002.

[Fin06] Olivier Finkel. Undecidable problems about timed automata. In Eugene Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 187–199. Springer-Verlag, September 2006.

[KT09] Moez Krichen and Stavros Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, June 2009.

[LMN04] Kim Guldstrand Larsen, Marius Mikučionis, and Brian Nielsen. Online testing of real-time systems using Uppaal. In Jens Grabowski and Brian Nielsen, editors, *Proceedings of the 4th International Workshop on Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *Lecture Notes in Computer Science*, pages 79–94. Springer-Verlag, September 2004.

[NS03] Brian Nielsen and Arne Skou. Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer*, 5(1):59–77, November 2003.

[Ram98] Solofo Ramangalahi. Strategies for comformance testing. Research Report 98-010, Max-Planck Institut für Informatik, May 1998.

[SVD01] Jan Springintveld, Frits Vaandrager, and Pedro R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, March 2001.

[Tre96] Jan Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29(1):49–79, 1996.

[Yan04] Mihalis Yannakakis. Testing, optimization, and games. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, Lecture Notes in Computer Science, pages 28–45. Springer-Verlag, 2004.

Appendix

We conduct here the proofs of the different claims. They are separated according to the corresponding parts of the article: testing framework, games, and test-case properties.

A Test framework

First, we prove the claim on the strong-connectivity of the reachable part of a specification semantics. This property is made quite intuitive by the introduction of always-reachable ζ-transitions. As it grounds our approach, we still provide a formal proof.

Proposition 7. *Let $\mathcal{S}$ be a specification with restarts. Then Reach$(\mathcal{T}_{\mathcal{S}})$ is strongly-connected.*

Proof. Let s be a configuration of $\mathcal{T}_{\mathcal{S}}$ reachable from $s_0^{\mathcal{S}}$. By hypothesis, there exists a finite partial execution starting in s which trace contains ζ. This trace leads to the configuration $s_0^{\mathcal{S}}$ hence any reachable configuration of $\mathcal{T}_{\mathcal{S}}$ is reachable from s, and we conclude that the reachable part of $\mathcal{T}_{\mathcal{S}}$ is strongly-connected. □

We now prove some properties of the product between specifications and test purposes.

Proposition A.1. *Let $\mathcal{S}$ be a specification and $\mathcal{TP}$ a test purpose on this specification. Then*

$$\mathsf{Seq}(\mathcal{S} \times \mathcal{TP}) = \mathsf{Seq}(\mathcal{S})$$

Proof. It suffices to note that the set of sequences of the product of two OTAIOs is the intersection of the sequences of the two original OTAIOs [BSJK15], so that $\mathsf{Seq}(\mathcal{S} \times \mathcal{TP}) = \mathsf{Seq}(\mathcal{S}) \cap \mathsf{Seq}(\mathcal{TP})$; we also note that $\mathcal{TP}$ is complete, and hence it accepts any sequence. □

By projection on traces, we immediately get:

Corollary A.2. *Let $\mathcal{S}$ be a specification, and $\mathcal{TP}$ a test purpose. $\mathcal{S}$ and $\mathcal{S} \times \mathcal{TP}$ are trace-equivalent.*

Corollary A.3. *Let $\mathcal{S}$ be a specification, $\mathcal{TP}$ a test purpose and $\mathcal{T}_{\mathcal{S}\times\mathcal{TP}}$ its associated timed transition system. The reachable part of $\mathcal{T}_{\mathcal{S}\times\mathcal{TP}}$ is strongly-connected.*

Proof. This proof is derived from the proof of Prop. 7. Although they are really close, we have to do it again as the product does not ensure any relation on the semantics.

Let $((l^1, l^2), v)$ be a reachable configuration of $\mathcal{T}_{\mathcal{S}\times\mathcal{TP}}$. There exists a finite partial execution starting in (l^1, v) whose trace contains ζ, and thus by Corollary A.2 there exists a finite partial execution starting in $((l^1, l^2), v)$ whose trace

contains ζ. Hence this transition leads to the configuration $s_0 = (l_0^{\mathcal{S}\times\mathcal{TP}}, \overline{0})$. It comes that there exists a finite partial execution from $((l^1, l^2), v)$ to s_0. Hence any reachable configuration of $\mathcal{T}_{\mathcal{S}\times\mathcal{TP}}$ is reachable from $((l^1, l^2), v)$. It can then be concluded that the reachable part of $\mathcal{T}_{\mathcal{S}}$ is strongly-connected. □

The following properties concern the objective-centered tester. They mainly amount to proving that the objective-centered tester keeps the interesting properties of the product, and that its traces are related to those of the previous automata.

Proposition A.4. *Let $\mathcal{DP}$ be the exact determinization of the product $\mathcal{P}$ between a specification and a test purpose, and $\mathcal{OT}$ its associated objective-centered tester. Then*

$$\mathit{Traces}(\mathcal{DP}) = \mathit{Traces}_{\mathit{conf}}(\mathcal{OT}).$$

Proof. An execution is in $\mathsf{Ex}_{\mathsf{conf}}(\mathcal{OT})$ if it avoids the verdict ***Fail***. This amounts to avoiding the location Fail and respecting the invariants of $\mathcal{DP}$. By construction of $\mathcal{OT}$, this corresponds exactly to the runs of $\mathcal{DP}$. □

Lemma A.5. *Given an objective-centered tester $\mathcal{OT}$, we have*

$$\mathit{Traces}_{\mathit{conf}}(\mathcal{OT}) \cap \mathit{Traces}(\mathit{Ex}_{\mathit{fail}}(\mathcal{OT})) = \emptyset.$$

Proof. Let $\rho \in \mathsf{Ex}_{\mathsf{fail}}(\mathcal{OT})$. Consider the longest prefix ρ' of ρ that does not reach ***Fail***, and let e be the transition taken after ρ' in ρ. Two cases should be considered:

- If e is a delay transition, then it violates the invariant of the location in $\mathcal{DP}$. By determinism of $\mathcal{DP}$, $\mathcal{DP}$ after $\mathsf{Trace}(\rho')$ is a singleton. Hence the same delay is not available after ρ in $\mathcal{DP}$.
- If $\mathsf{act}(e) \in \Sigma_!$ then this output is not specified in the current location of $\mathcal{DP}$. By determinism of $\mathcal{DP}$, $\mathcal{DP}$ after $\mathsf{Trace}(\rho')$ is a singleton. Hence transition e is not possible after ρ in $\mathcal{DP}$.

In both cases $\mathsf{Trace}(\rho) \notin \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$. As this holds for any run of $\mathsf{Ex}_{\mathsf{fail}}(\mathcal{OT})$ we have the desired property. □

Lemma A.6. *Let $\mathcal{OT}$ be an objective-centered tester. For any location l in $L^{\mathcal{OT}} \setminus \{\mathit{Fail}\}$ there exists a finite partial execution $\rho \in \mathit{pEx}(\mathcal{OT})$ starting in l and containing a ζ.*

Proof. The same result holds from any state in $\mathcal{S}$, and $\mathsf{Traces}(\mathcal{S}) = \mathsf{Traces}(\mathcal{S}\times\mathcal{TP})$. The result follows by exact determinizability of the product $\mathcal{S} \times \mathcal{TP}$. □

This lemma is the reason why our method assumes exact determinizability, as we can't ensure in general that the restart will remain reachable: if determinization is approximated, some traces might be lost.

Corollary A.7. *Let $\mathcal{OT}$ be an objective-centered tester and $\mathcal{T}^{\mathcal{OT}}$ its associated timed transition system. Then $\mathit{Reach}(\mathcal{T}^{\mathcal{OT}}) \setminus$ **Fail** is strongly-connected.*

Proof. The proof is the same as the one of Prop. 7, using Lemma A.6. □

Lemma A.8. *For a repeatedly-observable specification $\mathcal{S}$, $\mathsf{Reach}(\mathcal{OT}) \setminus \boldsymbol{Fail}$ is repeatedly-observable.*

Proof. We know that $\mathsf{Traces}(\mathcal{S}) = \mathsf{Traces}(\mathcal{P})$, hence for all $\sigma \in \mathsf{Traces}(\mathcal{P})$, $\mathsf{out}(\mathcal{P} \text{ after } \sigma) = \mathsf{out}(\mathcal{S} \text{ after } \sigma)$. As $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$ by assumption, we also know that for all $\sigma \in \mathsf{Traces}(\mathcal{DP})$, $\mathsf{out}(\mathcal{P} \text{ after } \sigma) \subseteq \mathsf{out}(\mathcal{DP} \text{ after } \sigma)$. It comes

$$\forall \sigma \in \mathsf{Traces}(\mathcal{OT}),\ \mathsf{out}(\mathcal{S} \text{ after } \sigma) \subseteq \mathsf{out}(\mathcal{OT} \text{ after } \sigma)$$

as $\mathcal{OT}$ only adds traces to $\mathcal{DP}$. Hence for all $s \in \mathsf{Reach}(S^{\mathcal{OT}}) \setminus \boldsymbol{Fail}$, there exists $\mu \in \mathsf{Seq}(\mathcal{OT})$ s.t. $s \xrightarrow{\mu} \wedge\mathsf{Trace}(\mu) \notin \mathbb{R}_{\geq 0}$. Indeed, there exists $\sigma \in \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ such that $s = \mathcal{OT} \text{ after } \sigma$ (as $\mathcal{OT}$ is deterministic outside of $\boldsymbol{Fail}$) and for $s' \in \mathcal{S} \text{ after } \sigma$, there exists μ' such that $s' \xrightarrow{\mu'} \wedge\mathsf{Trace}(\mu') \notin \mathbb{R}_{\geq 0}$. It suffices to take $\mu \in \mathsf{pSeq}(\mathcal{OT})$ such that $\mathsf{Trace}(\mu) = \mathsf{Trace}(\mu')$, and by the previous trace-inclusion property, such a trace exists. □

B Games

In this part the previous propositions are used to ensure that the sequence $(W_i^j)_{i,j}$ defines a partial order, and covers the reachable part of our game.

Proposition 14. *There exists $i, j \in \mathbb{N}$ such that $\mathsf{Reach}(\mathcal{G}^{\mathcal{OT}}) \setminus \boldsymbol{Fail} \subseteq W_i^j$.*

Proof. Let $s \in \mathsf{Reach}(\mathcal{G}) \setminus \boldsymbol{Fail}$ be a reachable configuration. Since i) $\boldsymbol{Pass}$ is reachable from $s_0^{\mathcal{T}}$ (by hypothesis); ii) there is a path from s back to the initial configuration (Corollary A.7), then $\boldsymbol{Pass}$ is reachable from s. Moreover, there is such a path with length bounded by the number of regions.

For each $s \in \mathsf{Reach}(\mathcal{G}) \setminus \boldsymbol{Fail}$, we fix a finite path to $\boldsymbol{Pass}$, and reason by induction on the length n of this path in order to prove that $s \in W_0^n$:

- Case $n = 0$: in this case $s \in \boldsymbol{Pass} = W_0^0$
- Inductive case: we assume that the result holds for n, and take s with a path to $\boldsymbol{Pass}$ of length $n+1$. Then $s \xrightarrow{e} s'$ for some e with $\mathsf{act}(e) \in \Gamma$, and there is a path from s' to $\boldsymbol{Pass}$ of length at most n, so that $s' \in W_0^n$. Hence in the worst case $s \in W_0^{n+1}$.

This proves our result. □

Proposition 16. *$\sqsubseteq$ is a partial order on $\mathsf{Reach}(\mathcal{G}^{\mathcal{OT}}) \setminus \boldsymbol{Fail}$.*

Proof. $\sqsubseteq$ is an order because it directly inherits the properties of $\leq_{\mathbb{N}^2}$. It is not total because several configurations can have the same rank. □

The following lemma is the key allowing to ensure that rank-lowering strategies are winning on fair executions.

Lemma 18. *If $\mathcal{OT}$ is repeatedly-observable, then for all $\rho = ((s_i, \gamma_i, s_{i+1}))_{i\in\mathbb{N}} \in \mathsf{Ex}(\mathcal{G})$ ending with an infinite sequence of delays, we have*

$$\rho \in \mathsf{Ex}_{\mathsf{fail}}(\mathcal{G}) \vee \exists e \in E^{\mathcal{G}},\ \overset{\infty}{\exists}\ i \in \mathbb{N}, e \in \mathsf{enab}(s_i).$$

Proof. We show this lemma by contradiction. Assume that for some $\rho \in \mathsf{Ex}(\mathcal{G})$, we have

$$\rho \notin \mathsf{Ex}_{\mathsf{fail}}(\mathcal{G}) \wedge \forall e \in E^{\mathcal{G}},\ \overset{\infty}{\forall}\ i \in \mathbb{N},\ e \notin \mathsf{enab}(s_i).$$

Let $\rho_{\max}$ be the shortest prefix such that no transition is enabled after this prefix along ρ (it exists because E is finite and there is only a finite number of these prefixes per element of E). Consider any prefix ρ' of ρ strictly containing $\rho_{\max}$; there is no partial sequence μ such that $\mathsf{last}(\rho') \xrightarrow{\mu}$ and $\mathsf{Trace}(\mu) \notin \mathbb{R}_{\geq 0}$, as there is no time successor of $\mathsf{last}(\rho')$ with an enabled transition. This contradicts the repeated-observability of $\mathcal{OT}$ out of Fail (as $\mathcal{G}$ and $\mathcal{OT}$ are the same automaton). □

Proposition 21. *Rank-lowering strategies are winning on $\mathsf{Fair}(\mathcal{G})$ (i.e., all fair outcomes are winning).*

In order to make this proof, we reason on regions. For this purpose we extend to regions the notions of executions and enabled transitions. We furthermore note that a region is included in any W_i^j it intersects. We first prove the following lemma:

Lemma B.9. *Let $\rho \in \mathsf{Fair}(\mathcal{G})$ be a fair execution and $reg \in \mathsf{Inf}(\rho)$. For any prefix ν of ρ ending in reg, and any rank-lowering strategy f, noting $f(\nu) = (\delta, a)$, we have $\nu \xrightarrow{\delta} s \in reg'$ and $reg' \in \mathsf{Inf}(\rho)$.*

Proof. By definition of a rank-lowering strategy, δ is a possible delay after $\mathsf{last}(\nu)$. Hence there exists s and reg' such that $\nu \xrightarrow{\delta} s \in reg'$. By definition of rank-lowering strategies, it is always the same reg' for each ν. If $reg = reg'$ then we have our result. Otherwise, by definition of outcomes, there is no transition labelled with a controllable transition leaving reg, and by our fairness assumption, there exists a (strict) time successor $reg \xrightarrow{t} reg''$ of reg such that $reg'' \in \mathsf{Inf}(\rho)$. If $reg'' = reg'$ we have our result; otherwise, by defintion of outcomes, reg'' is a time predecessor of reg', and applying the same arguments to reg'' will create an induction (as by definition of rank-lowering strategies, the strategy after going from ν to reg'' is to delay to reg'). As there is only finitely-many regions between reg and reg', we have our result by the induction principle. □

With this lemma, the proof of the main proposition is made easier.

Proof (of Prop. 21). Let $\mathcal{T} = (S, s_0, \Gamma, \rightarrow_{\mathcal{T}})$ be the timed transition system associated with $\mathcal{G} = (L, l_0, \Sigma_u \uplus \Sigma_c, X, I, E)$. Let f be a rank-lowering strategy. We want to prove that $\mathsf{Outcome}(s_0, f) \cap \mathsf{Fair}(\mathcal{G}) \subseteq \mathsf{Win}(\mathcal{G})$. We proceed by contradiction.

Suppose there exists an infinite run $\rho \in \mathsf{Outcome}(s_0, f) \cap \mathsf{Fair}(\mathcal{G})$ such that $\rho \notin \mathsf{Win}(\mathcal{G})$. We denote $r_{\min}$ the minimal rank obtained in $\mathsf{Inf}(\rho)$ and $reg \in \mathsf{Inf}(\rho)$ such that $r(reg) = r_{\min}$. For each prefix ν of ρ ending in a configuration $s = (l, v) \in reg$, we let $(\delta_\nu, a_\nu) = f(\nu)$. We consider three cases, following the definition of rank-lowering strategies:

- Assume $\mathsf{last}(\nu) \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_c}(W^-(\mathsf{last}(\nu))))$ and $a_\nu \in \Sigma_c$. By Lemma B.9, there exists $reg' \in \mathsf{Inf}(\rho)$ such that $\nu \xrightarrow{\delta_\nu} s_\nu$ and $s_\nu \in reg'$. Hence there exists a transition e such that $\mathsf{act}(e) = a_\nu$. The system cannot delay more by definition of $\mathsf{Outcome}(s_0, f)$ and by fairness, $reg' \xrightarrow{e} reg''$ and $reg'' \in W^-(\mathsf{last}(\nu)) \cap \mathsf{Inf}(\rho)$, which contradicts the minimality of $r_{\min}$.
- Assume $\mathsf{last}(\nu) \in \mathsf{tPred}(W^-(\mathsf{last}(\nu)))$ and $a_\nu = \bot$. By Lemma B.9, there exists $reg' \in \mathsf{Inf}(\rho)$ such that $\nu \xrightarrow{\delta_\nu} s_\nu$ and $s_\nu \in reg'$. By definition of the case, $reg' \in W^-(\mathsf{last}(\nu))$, thus the minimality of $r_{\min}$ is contradicted.
- We finally consider the last case: as the duration in this one is maximal, we have $\mathsf{last}(\nu) \notin \mathsf{tPred}(W^-(\mathsf{last}(\nu)) \cup \mathsf{Pred}_{\Sigma_c}(W^-(\mathsf{last}(\nu))))$ and there are two cases to consider:
  - if $(W^-(\mathsf{last}(\nu))$ is undefined, then $r(\mathsf{last}(\rho)) = (0, 0)$ and ρ is winning, which is a contradiction;
  - otherwise, by Proposition 14, $\mathsf{last}(\nu) \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_u}(W^-(\mathsf{last}(\nu))))$ (this corresponds either to ftPred or to a W_0^{j+1}). Furthermore $a_\nu = \bot$ and δ_ν leads to the maximal delay successor region, which we call reg'. By Lemma B.9, $reg' \in \mathsf{Inf}(\rho)$. Hence, by definition of Inf, all regions between reg and reg' are in $\mathsf{Inf}(\rho)$. In particular, a region $reg'' \in \mathsf{Inf}(\rho)$ such that $reg'' \xrightarrow{e} reg''' \in W^-(\mathsf{last}(\rho))$ and $\mathsf{act}(e) \in \Sigma_u$ exists by definition of the case. Hence by fairness, $reg''' \in \mathsf{Inf}(\rho)$ and the minimality of $r_{\min}$ is contradicted. □

C Test case properties

Proposition 24. *The test-case generation method is* sound*: for any specification $\mathcal{S}$, it holds*

$$\forall \mathcal{I} \in \mathcal{I}(\mathcal{S}),\ \forall (\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S}),\ (\mathcal{I}\ \textit{fails}\ (\mathcal{G}, f) \Rightarrow \neg(\mathcal{I}\ \textit{tioco}\ \mathcal{S})).$$

Proof. Let $\mathcal{S}$ be a specification, $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ and $(\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S})$. Suppose that $\mathcal{I}$ fails $(\mathcal{G}, f)$, we will prove that $\neg(\mathcal{I}\ \mathsf{tioco}\ \mathcal{S})$.

Since $\mathcal{I}$ fails $(\mathcal{G}, f)$, there is a finite run ρ of $\mathsf{Behaviour}(\mathcal{G}, f, I)$ such that $last(\rho) \in \boldsymbol{Fail} \times S^{\mathcal{I}}$ and it is the first configuration of ρ in this set. Let $\sigma = \mathsf{Trace}(\rho)$. By construction of $\boldsymbol{Fail}$, either $\sigma = \sigma' \cdot \delta$ (if the configuration of $\boldsymbol{Fail}$ reached corresponds to a faulty invariant) or $\sigma = \sigma' \cdot a$ with $a \in \Sigma_!$ (and Fail is reached). In both cases $\mathsf{out}(\mathcal{I}\ \mathsf{after}\ \sigma') \not\subseteq \mathsf{out}(\mathcal{DP}\ \mathsf{after}\ \sigma')$, and by definition $\neg(I\ \mathsf{tioco}\ \mathcal{DP})$.

As $\mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{DP})$ by exact-determinizability hypothesis, $\neg(I\ \mathsf{tioco}\ \mathcal{P})$. Finally, as $\mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{S})$, we have $\neg(I\ \mathsf{tioco}\ \mathcal{S})$, which concludes the proof. □

Remark 1. Note that this proof is more general than the property, as it does not rely on the strategy. It hence proves the property for any strategy f and not only for rank-lowering ones. The key reason lies in fact in the structure of $\mathcal{G}$, and ensures that any run reaching ***Fail*** has the correct form.

Proposition 26. *The test generation method is* strict*: given a specification $\mathcal{S}$,*

$$\forall \mathcal{I} \in \mathcal{I}(\mathcal{S}),\ \forall (\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S}),\ \neg(\mathit{Behaviour}(\mathcal{G}, f, \mathcal{I})\ \mathit{tioco}\ \mathcal{S}) \Rightarrow \mathcal{I}\ \mathit{fails}\ (\mathcal{G}, f)$$

Proof. Let $\mathcal{S}$ be a specification, $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ and $(\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S})$. Suppose that $\neg(\mathsf{Behaviour}(\mathcal{G}, f, \mathcal{I})\ \mathsf{tioco}\ \mathcal{S})$. We want to show that $\mathcal{I}$ fails $(\mathcal{G}, f)$. By definition of $\neg(\mathsf{Behaviour}(\mathcal{G}, f, \mathcal{I})\ \mathsf{tioco}\ \mathcal{S})$, there exist $\sigma \in \mathsf{Traces}(\mathcal{S})$ and

$$a \in \mathsf{out}(\mathsf{Behaviour}(\mathcal{G}, f, \mathcal{I})\ \mathsf{after}\ \sigma) \setminus \mathsf{out}(\mathcal{S}\ \mathsf{after}\ \sigma)$$

Since $\mathcal{DP}$ is an exact determinization of $\mathcal{P}$ we have the following equalities: $\mathsf{Traces}(\mathcal{S}) = \mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$. Since $a \in \mathbb{R}_{\geq 0} \cup \Sigma_!$, $\sigma \cdot a \in \mathsf{Traces}(\mathcal{OT})$ as invariants have been removed, and the automaton has been completed on $\Sigma_!$ with transitions to Fail). Hence $\sigma \cdot a \in \mathsf{Traces}(\mathsf{Ex}_{\mathsf{fail}}(\mathcal{OT}))$. Thus, for $\rho \in \mathsf{Behaviour}(\mathcal{G}, f, \mathcal{I})$ such that $\mathsf{Trace}(\rho) = \sigma \cdot a$, $\mathsf{last}(\rho) \in \boldsymbol{\mathit{Fail}}$ and $\mathcal{I}$ fails $(\mathcal{G}, f)$. □

Note that once again, the properties of the strategy are not used.

Proposition 27. *The test case generation method is* precise*: for any specification $\mathcal{S}$ and test purpose $\mathcal{TP}$ it can be stated that*

$$\forall (\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S}, \mathcal{TP}), \forall \sigma \in \mathit{Traces}(\mathit{Outcome}(s_0^{\mathcal{G}}, f)),$$
$$\mathcal{G}\ \mathit{after}\ \sigma \in \boldsymbol{\mathit{Pass}} \ \Leftrightarrow\ (\sigma \in \mathit{Traces}(\mathcal{S}) \wedge \mathcal{TP}\ \mathit{after}\ \sigma \cap \mathit{Accept}^{\mathcal{TP}} \neq \emptyset)$$

Proof. Let σ be in $\mathsf{Traces}(\mathsf{Outcome}(s_0^{\mathcal{G}}, f))$. Then $\mathcal{G}$ after $\sigma \in \boldsymbol{\mathit{Pass}}$ if, and only if, the run ρ such that $\mathsf{Trace}(\rho) = \sigma$ (which is unique by determinism of $\mathcal{G}$ outside ***Fail***) is such that $\mathsf{last}(\rho) \in \boldsymbol{\mathit{Pass}}$, *i.e.* $\rho \in \mathsf{Ex}(\mathcal{DP})$ and $\mathsf{last}(\rho) \in \mathsf{Accept}^{\mathcal{DP}}$. Hence $\mathcal{DP}$ after $\sigma \in \mathsf{Accept}^{\mathcal{DP}}$ and as the determinization is exact, $\sigma \in \mathsf{Traces}(\mathcal{P})$ and $\mathcal{P}$ after $\sigma \in \mathsf{Accept}^{\mathcal{P}}$, which gives by definition $\sigma \in \mathsf{Traces}(\mathcal{S}) \wedge \mathcal{TP}$ after $\sigma \cap \mathsf{Accept}^{\mathcal{TP}} \neq \emptyset$. □

The proof uses only properties of the game, and once more does not rely on the precise strategy used.

Proposition 28. *The test generation method is* exhaustive*: for any exactly determinizable specification $\mathcal{S}$ and any implementation $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ making fair runs*

$$\neg(\mathcal{I}\ \mathit{tioco}\ \ \mathcal{S}) \Rightarrow \exists (\mathcal{G}, f) \in \mathcal{TC}(\mathcal{S}), \mathcal{I}\ \mathit{fails}\ (\mathcal{G}, f).$$

Proof. Let $\mathcal{S}$ be a specification, and $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ a non-conformant implementation. By definition of $\neg(\mathcal{I}\ \mathsf{tioco}\ \mathcal{S})$, there exists $\sigma \in \mathsf{Traces}(\mathcal{S})$ and $a \in \mathbb{R}_{\geq 0} \cup \Sigma_!$ such that $a \in \mathsf{out}(\mathcal{I}\ \mathsf{after}\ \sigma)$ and $a \notin \mathsf{out}(\mathcal{S}\ \mathsf{after}\ \sigma)$. As $\mathcal{S}$ is repeatedly-observable,

there exists $\delta \in \mathbb{R}_{\geq 0}$ and $b \in \Sigma^{\mathcal{S}}_{\mathsf{obs}}$ such that $\sigma \cdot \delta \cdot b \in \mathsf{Traces}(\mathcal{S})$. Because $\mathcal{S}$ is also non-blocking, if a is a delay, we can take $b \in \Sigma^{\mathcal{S}}_{!}$. Indeed, otherwise there would be no trace controlled by the implementation for any finite time (say, for time a).

It is possible to build a test purpose $\mathcal{TP}$ that accepts exactly the trace $\sigma \cdot \delta \cdot b$. It suffices to send every transition that is not part of this trace to a sink location. As $\sigma \cdot \delta \cdot b \in \mathsf{Traces}(\mathcal{S})$ it is also a trace of the product $\mathcal{P} = \mathcal{S} \times \mathcal{TP}$. As $\mathcal{S}$ is exactly determinizable and $\mathcal{TP}$ is deterministic, $\mathcal{P}$ is exactly determinizable by allowing enough resources to $\mathcal{DP}$. We thus obtain $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$ and $\sigma \cdot \delta \cdot b \in \mathsf{Traces}(\mathcal{DP})$. Hence, the minimal elements of $\boldsymbol{Pass}$ are $\mathcal{OT}$ after $\sigma \cdot \delta \cdot b$.

From $\mathcal{OT}$ a test case $(\mathcal{G}, f)$ can be built, with f a rank-lowering strategy. By assumption, the implementation is playing fair runs, hence f is winning. So there exists $\rho \in \mathsf{Outcome}(s^{\mathcal{G}}_0, f)$ such that $\mathsf{Trace}(\rho) = \sigma \cdot \delta \cdot b$, and thus there exists $\rho' \in \mathsf{Outcome}(s^{\mathcal{G}}_0, f)$ such that $\mathsf{Trace}(\rho') = \sigma$. By assumption, $\sigma \cdot a \in \mathsf{Traces}(\mathcal{I})$, and depending on the nature of a:

- If $a \in \Sigma_!$ then $\sigma \cdot a \in \mathsf{Outcome}(s^{\mathcal{G}}_0, f)$ as $\mathcal{G}$ is complete on $\Sigma_!$. Hence $\sigma \cdot a \in \mathsf{Behaviour}(\mathcal{G}, f, \mathcal{I})$ and as $\sigma \cdot a \notin \mathsf{Traces}(\mathcal{S})$ and the determinization is exact, $\sigma \cdot a \notin \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ and $\mathcal{G}$ after $\sigma \cdot a \in \boldsymbol{Fail}$. Hence $\mathcal{I}$ fails $(\mathcal{G}, f)$.
- If a is a delay, then $a > \delta$, and $b \in \Sigma_!$. As b is controlled by the implementation, and there is no invariant in $\mathcal{G}$, $\sigma \cdot a \in \mathsf{Outcome}(s^{\mathcal{G}}_0, f)$. Hence $\sigma \cdot a \in \mathsf{Behaviour}(\mathcal{G}, f, \mathcal{I})$ and as $\sigma \cdot a \notin \mathsf{Traces}(\mathcal{S})$ and the determinization is exact, $\sigma \cdot a \notin \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ and $\mathcal{G}$ after $\sigma \cdot a \in \boldsymbol{Fail}$. Hence $\mathcal{I}$ fails $(\mathcal{G}, f)$.

□