Timed Testing under Partial Observability

David, Alexandre; Larsen, Kim Guldstrand; Li, Shuhao; Nielsen, Brian

Published in:
Proceedings of 2009 International Conference on Software Testing Verification and Validation

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Timed Testing under Partial Observability

Alexandre David, Kim G. Larsen, Shuhao Li, Brian Nielsen
Center for Embedded Software Systems (CISS)
Aalborg University
DK-9220 Aalborg, Denmark
{adavid, kgl, li, bnielsen}@cs.aau.dk

Abstract

This paper studies the problem of model-based testing of real-time systems that are only partially observable. We model the System Under Test (SUT) using Timed Game Automata (TGA) which has internal actions, uncontrollable outputs and timing uncertainty of outputs. We define the partial observability of SUT using a set of predicates over the TGA state space, and specify the test purposes in Computation Tree Logic (CTL) formulas. A recently developed partially observable timed game solver is used to generate winning strategies, which are used as test cases. We propose a conformance testing framework, define a partial observation-based conformance relation, present the test execution algorithms, and prove the soundness and completeness of this test method (i.e., a detected error really violates the conformance relation; and if the SUT violates the test purpose, then a test case can be generated to detect this violation). Experiments on some non-trivial examples show that this method yields encouraging results.

1. Introduction

Timed automata (TA) [2] has been widely adopted as formalisms for real-time systems that are safety-/mission-/economic- critical. A considerable proportion of the real-time testing work [12, 20, 14, 13, 19, 17, 6, 15] use TA or timed transition systems to model the systems. To enable conformance testing, these work build their implementation relations on top of e.g. trace equivalence [12, 20] or the ioco conformance relations [14, 13, 17, 6, 15]. To steer the testing towards certain test purposes or test coverage, some of these methods need (to be enhanced with) the assumption of *full observability* (a.k.a. *perfect information*), i.e., at any time, the tester knows precisely what state or configuration (TA locations, clocks, and data variables) the system under test (SUT) is in, or she can uniquely infer one such state or configuration by observing an externally observable timed input/output action sequence on the tester/SUT interface. The full observability assumption paves way to accurately driving, monitoring test executions and issuing test verdicts. However, in practice this assumption is hardly realistic. On one hand, if the SUT consists of several interacting components, then the tester may observe neither the internally coupling inputs/outputs, nor the internal state changes that are caused by those internal actions. On the other hand, if the tester has only limited precision sensors to measure the SUT, she might not tell which exact state the SUT is in or might not precisely observe a timed input/output sequence. Environment noises and external interferences could also affect the observations.

In this paper we consider the problem of testing timed systems that are only *partially observable* (or, with *imperfect information*). One way to characterize partial observability is to assume that only a proper subset of those outputs from the SUT to the tester (or environment, or user) can be observed, and/or only a proper subset of the system clocks can be read by the tester [5]. In another way, partial observability is characterized in terms of a finite number of possible observations to be made on the SUT states (configurations) [8]. This paper follows the latter approach, which has mature algorithms and tool supports.

We use the Smart Light Controller [13] as a running example. Fig. 1 and 2 give the TA models of the light control program and its user, respectively. The user interacts with the light by touching a touch-sensitive pad. In Fig. 1, there are four brightness levels (in ascending order) for the light: Off, Dim1, Dim2 and Bright. The light is initially in location Off. If the pad is touched at an appropriate time ($x \geq 2$), the light will go to location L1 where within 2 time units it will non-deterministically go to Bright or go to Dim1. At location Dim1, a *touch*? input at appropriate time can bring the TA to Dim2. The light can automatically go to Dim2 at any time. If it stays in Dim1 for a period of time ($x \geq 3$), it can automatically go Off at any time after that period. These are internal transitions which need no synchronization with the user.

The dashed lines in Fig. 1 denote transitions that are controlled by the light rather than by the user. Note that in Dim1, Dim2, L1 and L2: (1) the internal transitions can autonomously occur when the conditions (if any) are satisfied, and their occurrences cannot be observed by the user; and (2) the light itself decides whether to make an output or an internal transition, and if yes, which output or internal transition will be made, and (3) when to make that output or internal transition. These three characteristics are called *internal actions*, *uncontrollable actions*, and *timing uncertainty of uncontrollable actions*, respectively. A TA with these characteristics is a liberal specification model which can be refined into a family of similar but different implementations. If the tester offers an input stimulus, different implementations could produce different responses.

In [10] we view testing of a TA-modeled timed system as playing a timed game under full observability, where the tester acts as a game player, and the SUT acts as the game opponent. The test purposes are given in extended reachability or safety CTL formulas, which describes the functional or timing properties to be satisfied by the SUT. For example, "control: A$\langle\rangle$ Bright" means that the tester can manage to guarantee that the Bright location will be reached, regardless of whatever reaction uncertainties there might be with the SUT. We check the satisfiability of the test purposes using an existing time game solver UPPAAL-TIGA [7, 3]. If the outcome is positive, the tool can synthesize a *state-based* winning strategy for the tester, which ensures enforcing the test purposes by providing step-by-step guidance to the tester, e.g., "if the SUT is in states $\langle$Dim1, $1 \leq x < 3\rangle$, the tester should offer a *touch*? to the SUT; if in states $\langle$Dim1, $x < 1\rangle$, the tester should just wait (stay quiet) there". In this way the tester will certainly win the game (by finally arriving in a "good" state, or by constantly avoiding a "bad" state). In an off-line testing manner, the generated strategies are used as test cases to test a family of different implementations.

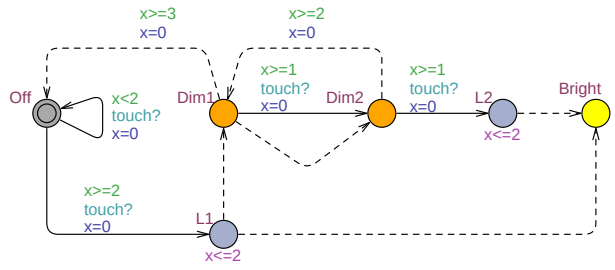

Figure 1. TA for the light (SUT).

If the SUT is only *partially observable*, say, in this Light Controller example the tester can observe [1]:

(1) whether or not the SUT is "off" (i.e., in Off), and

[1]In practice, the observable predicates of (1)-(3) can be implemented by probing/instrumenting the SUT with some light sensors or software-defined location reporters; the assumption in predicate (2) is reasonable if the difference between these two brightness levels is too small to be discerned by the light sensors.

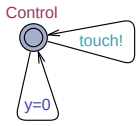

Figure 2. TA for the user (tester).

(2) whether or not the SUT is "dim" (i.e., in Dim1 or Dim2, but not exactly in which one), and

(3) whether or not the SUT is "bright" (i.e., in Bright),

then the method in [7] [10] no longer applies. On one hand, synthesis of state-based strategies is based on the full observability assumption. On the other hand, the tester can not use (execute) a pre-computed strategy tree under partial observability. Consider that the tester feeds the SUT with a timed input preamble $2 \cdot touch? \cdot 2$ at the initial state of Fig. 1. She may get the observation "dim" signaling that either location Dim1 or location Dim2 has been reached. But since she does not know the exact location, she has no idea which next move to take according to the strategy.

Now our problem is that *under partial observability, given a test purpose, is it possible to synthesize an observation-based winning strategy for the tester to ensure that the Bright location is always reachable, regardless of whatever reaction uncertainties in the model*? And if so, how can we *use this strategy to test whether a concrete implementation IMP also respects the test purpose*?

1.1 Related work

Model-based testing can be comprehensive testing [12, 20, 13, 19, 17, 6, 15] or targeted testing [14, 13]. We follow the latter approach, i.e., we use test purposes to test whether the system satisfies some functional/timing properties.

Much existing work on timed testing use TA to model the systems in question. For the sake of testability [20] the TA is restricted to be *predictable*, i.e., it should be deterministic (or determinizable), output-urgent and have isolated outputs [2] [20, 12, 13]. This predictability leads to full observability, and thus favors well-steered test executions. But from a model-based development point of view, these TA models

[2]Given a TA (L, l_0, Act, X, E, Inv) where L is the location set, l_0 the initial location, $Act = (Act_I \cup Act_O)$ the set of input and output actions, and X, E, Inv the set of clocks, edges, and invariants, respectively. Let $(S, s_0, Act, \rightarrow)$ be its underlying timed labeled transition system. We say that the TA is *deterministic* if $\forall s \in S.\forall \alpha \in Act.((s \xrightarrow{\alpha} s') \wedge (s \xrightarrow{\alpha} s'') \Rightarrow (s' = s''))$. The TA is *output-urgent* if $\forall s \in S.\forall \alpha \in Act_O.((s \xrightarrow{\alpha}) \Rightarrow \forall d \in \mathbb{R}_{>0}.(s \not\xrightarrow{d}))$. The TA has *isolated outputs* if $\forall s \in S.\forall \alpha \in Act_O.\forall \beta \in Act.(((s \xrightarrow{\alpha}) \wedge (s \xrightarrow{\beta})) \Rightarrow (\alpha = \beta))$.

are too detailed and restricted to be suitable for early-stage modeling. In contrast, a liberal TA model will allow the implementor more freedom, will help the tester capture essential high-level design requirements rather than those less important implementation details, and a liberal TA model is usually more natural and more succinct than a detailed one.

By canceling the restrictions of isolated outputs and output-urgency, we get the Timed Game Automaton (TGA) [18], which has tester-controllable transitions drawn in solid lines and uncontrollable transitions in dashed lines (see Fig. 1). Given a network (group) of TGA and a test purpose in the form of reachability or safety property, there are efficient algorithm [7] and tool [3] to solve the timed game.

Game-theoretic approaches to untimed system testing have been discussed in [1, 22, 4]. In the timed case, winning strategies for a given test purpose have been used as tests for off-line black-box conformance testing [10]. Specifically, if the test purpose is not satisfied by the SPEC model, we can still possibly synthesize cooperative winning strategies, and use them to test the SUT against the test purpose as long as the SUT reacts to our test inputs in a desired manner [9].

The methods in [10, 9] require the full observability assumption. By allowing component interactions inside the SUT model and measurement inaccuracy, we get partial observability. More recently, it has been shown that by fixing the resources of the controller (i.e. a maximum number of clocks and a maximum allowed constant in guards), the timed control problems based on these TGA are decidable [5, 8], and it is possible to synthesize observation-based stuttering-invariant strategies for the tester [8]. This motivates us to investigate the possibility of using such strategies as tests for timed systems under partial observability.

1.2 Contributions

The main contributions of this paper include: (1) we apply game strategies to the context of model-based testing, and propose a framework of conformance testing of timed systems based on partial observations; (2) we define an observation-based conformance relation between the SPEC and the IMP, propose test execution algorithms based on this relation, and prove their soundness and completeness; (3) we conduct preliminary case studies of test generation using a prototype tool, and report the experimental results.

2 Timed control under partial observability

2.1 Partially observable time game

Given two TGA modeling the controller program in question (the controller, or player 1) and the system under control (the plant, or player 2), and given a control objective formulated as e.g. a reachability or safety property, then a timed control problem consists in finding a winning strategy for the controller such that the control objective will be enforced regardless of how the plant reacts.

For the time control problem to be decidable [5, 8], we assume that all clock values in the TGA are bounded by a natural number, say M.

Let X be a finite set of non-negative real-valued variables called *clocks*, then $\mathcal{C}(X,\mathsf{M})$ is the set of constraints generated by the grammar $\varphi ::= x \sim k \mid x-y \sim k \mid \varphi \wedge \varphi$, where $k \leq \mathsf{M} \in \mathbb{Z}_{\geq 0}$, $x, y \in X$ and $\sim \in \{<, \leq, =, \geq, >\}$, and $\mathcal{B}(X,\mathsf{M})$ is a subset of $\mathcal{C}(X,\mathsf{M})$ defined by $\varphi ::= \mathsf{true} \mid k_1 \leq x < k_2 \mid \varphi \wedge \varphi$, where $k_1 < k_2 \leq \mathsf{M} \in \mathbb{Z}_{\geq 0}$, and $x \in X$.

Definition 1 (**Timed Game Automaton** [18]). A *Timed Game Automaton* (TGA) is a tuple $A = (L, l_0, Act_c, Act_u, X, E, Inv)$ where

- L is a finite set of locations,
- $l_0 \in L$ is the initial location,
- Act_c and Act_u are disjoint sets of player 1 controllable (input) actions and player 2 controllable (output or internal) actions, respectively. Specifically, internal action $\tau \in Act_u$,
- X is a finite set of real-valued clocks,
- E is a finite set of transitions partitioned into player 1 controllable ones belonging to $L \times \mathcal{B}(X,\mathsf{M}) \times Act_c \times 2^X \times L$ and player 2 controllable ones belonging to $L \times \mathcal{C}(X,\mathsf{M}) \times Act_u \times 2^X \times L$. Specifically, $(l, g, \tau, r, l') \in L \times \mathcal{C}(X,\mathsf{M}) \times Act_u \times 2^X \times L$ is an internal transition,
- $Inv : L \to \mathcal{B}(X,\mathsf{M})$ associates to each location its invariant.

For a network of interacting TGA, we define their parallel composition in the usual manner. The behavior of a TGA can be described using a Timed Labeled Transition System (TLTS).

Definition 2 (**2-player Timed Labeled Transition System**, **2-player TLTS**). A *2-player TLTS* is a tuple $(S, s_0, Act_c, Act_u, \to)$ where S is an (infinite) set of semantic states, s_0 the initial state, Act_c and Act_u the player 1 and player 2 controllable actions, respectively, and $\to \subseteq S \times (Act_c \cup Act_u \cup \mathsf{R}_{>0}) \times S$ is the transition relation.

Given an M-bounded timed automaton, a *valuation* v of the clock variables is a mapping from X to an interval of real numbers $[0,\mathsf{M}]$, i.e., $v : X \to [0,\mathsf{M}]$. The set of all such valuations is denoted $[X \to [0,\mathsf{M}]]$ or $[0,\mathsf{M}]^X$. For $r \subseteq X$, we denote by $[r \to 0]v$ the new valuation which is obtained from v by assigning 0 to any $x \in r$. Let $d \in \mathsf{R}_{\geq 0}$, v be an M-valuation, then $v+d$ is a valuation $(v+d)(x) = v(x)+d$ if for all $x \in X$, $v(x) + d \leq \mathsf{M}$. For $g \in \mathcal{C}(X,\mathsf{M})$ and $v \in [X \to [0,\mathsf{M}]]$, we write $v \models g$ if v satisfies g.

Given TGA $A = (L, l_0, Act_c, Act_u, X, E, Inv)$, let $K \subseteq L$ and $\varphi \in \mathcal{B}(X, \mathsf{M})$, then (K, φ) is called an *observable predicate*. We use a finite set of observable predicates $P \subseteq 2^L \times \mathcal{B}(X, \mathsf{M})$ to observe a TGA. For instance, in Fig. 1 we can have P = {({Off}, true), ({Dim1, Dim2}, true), ({Bright}, true), $(L, 0 \leq y < 1)$} [3].

An observable predicate (K, φ) is true at a TGA semantic state $s = \langle l, v \rangle$ iff ($l \in K$ and $v \models \varphi$). An *observation* $o_{P,s}$ of the TGA with predicates P at state s is a valuation of all the predicates in P at s. Formally, $o_{P,s} : P \to \{\mathsf{true}, \mathsf{false}\}$. For instance, in Fig. 1 at semantic state $\langle \mathsf{Dim2}, (x = 2, y = 4) \rangle$ we have the observation $o_{P,s}$ such that $o_{P,s}((\{\mathsf{Off}\}, \mathsf{true})) = \mathsf{false}$, $o_{P,s}((\{\mathsf{Dim1}, \mathsf{Dim2}\}, \mathsf{true})) = \mathsf{true}$, $o_{P,s}((\{\mathsf{Bright}\}, \mathsf{true})) = \mathsf{false}$, and $o_{P,s}((L, 0 \leq y < 1)) = \mathsf{false}$. Let O_P be the set of all possible observations with P, then by definition we have $|O_P| \leq 2^{|P|}$. Each element in O_P corresponds to a set of TGA states that have the same truth value for each of the observable predicates, hence an equivalence class. Thus O_P defines a partition of the TGA state space.

For TGA A and a set of observable predicates P, we define a function $\gamma_P : O_P \to 2^{L \times [X \to [0, \mathsf{M}]]}$ to map each observation o to its class of equivalent TGA states

$$\gamma_P(o) = \Big\{ \langle l, v \rangle \Big| \bigwedge_{\{(K,\varphi) | o(K,\varphi) = \mathsf{true}\}} (\langle l, v \rangle \vDash (K, \varphi)) \ \wedge \bigwedge_{\{(K,\varphi) | o(K,\varphi) = \mathsf{false}\}} (\langle l, v \rangle \nvDash (K, \varphi)) \Big\}.$$

We define a function $\xi_P : (L \times [X \to [0, \mathsf{M}]]) \to O_P$ to make observation of a TGA state. Clearly, for $\langle l, v \rangle \in S$, if $\langle l, v \rangle \in \gamma_P(o)$, then we have $\xi_P(\langle l, v \rangle) = o$.

A TGA which is associated with a set of observable predicates is called a *partially observable* TGA, whose semantics is defined as a 2-player TLTS.

Definition 3 (**TGA semantics**). The semantics of a TGA $A = (L, l_0, Act_c, Act_u, X, E, Inv)$ associated with a set P of observable predicates is a 2-player TLTS $S_A = (S, s_0, Act_c, Act_u, \to)$ where:

- $S = \{\langle l, v \rangle | (l \in L) \wedge v \in (\mathsf{R}_{\geq 0} \cap [0, \mathsf{M}])^X \wedge (v \models Inv(l))\}$, and $s_0 = \langle l_0, \bar{0} \rangle \in S$;
- the transition relation $\to$ is composed of
  - discrete transitions: $\langle l, v \rangle \xrightarrow{a} \langle l', v' \rangle$ if for $\langle l, v \rangle, \langle l', v' \rangle \in S$, and $a \in (Act_c \cup Act_u)$, there exists $e = (l, a, g, r, l') \in E$.($v \models g$ and $v' = [r \to 0]v$ and $v' \models Inv(l')$),
  - time transitions: $\langle l, v \rangle \xrightarrow{d} \langle l, v + d \rangle$ if for $\langle l, v \rangle \in S$, $d \in \mathsf{R}_{>0}$, and ξ_P over A and P, $\forall d' \in [0, d].(u + d' \models Inv(l))$ and $\forall d'' \in [0, d).\xi_P(\langle l, v + d'' \rangle) = \xi_P(\langle l, v \rangle)$.

Definition 3 requires that during the period of a time transition, the current observation never changes. Therefore, a new observation occurs only as a consequence of a discrete transition, or at the rightmost point of a time delay.

Let $s \in S$, we define the set of possible actions or delays at s as $enable(s) = \{\sigma \in (Act_c \cup Act_u \cup \mathsf{R}_{>0}) | \exists s' \in S.((s, \sigma, s') \in \to)\}$.

2.2 OBSI winning strategy

We consider *competing* rather than turn-based games between a plant and its controller (the tester).Thus if a controller move and a plant move are enabled at the same time, the former one could always be preempted by the latter one.

An observation-based stuttering-invariant (OBSI) strategy guides the controller to play the game in *sessions* (*macro* steps), during each of which the observation remains unchanged. A macro step consists of a number of consecutive *micro* steps, each of which is caused either by a controller move or by a plant move:

- During each session of the controller/plant interaction, the controller sticks to either a controllable input $a_1 \in Act_c$, or *delay*, until a new observation occurs.
  - If controller sticks to a_1, then plant can either choose to delay (only if a_1 is not enabled), or do an arbitrary uncontrollable action $a_2 \in Act_u$.
  - If controller sticks to *delay*, then plant can either choose to delay, or do an arbitrary uncontrollable action $a_2 \in Act_u$.
- Once a new observation occurs, a next session of controller/plant interaction begins.

By "stick to" an input action we mean that the controller can arbitrarily repeat that move (and only that move) as long as it is enabled. By "stick to" *delay* we mean that the controller keeps on delaying (and can only delay).

Definition 4 (**Play**). Let $\langle l_i, v_i \rangle$ be a TGA state, $c_i \in (Act_c \cup \{delay\})$ be a controller chosen (or desired) move, and $\sigma_i \in (Act_c \cup Act_u \cup \mathsf{R}_{>0})$ be the actually occurred transition, $i = 0, 1, \ldots, n$. A *play* is a sequence $\rho = \langle l_0, v_0 \rangle c_0 \sigma_0 \langle l_1, v_1 \rangle c_1 \sigma_1 \ldots \langle l_n, v_n \rangle c_n \sigma_n \ldots$, such that for all $i \geq 0, \langle l_i, v_i \rangle \xrightarrow{\sigma_i} \langle l_{i+1}, v_{i+1} \rangle$ and

- either ($\sigma_i = c_i \in Act_c$, or $\sigma_i \in Act_u$), or ($\sigma_i \in \mathsf{R}_{>0}$ if $\forall 0 \leq t < \sigma_i.c_i \notin enable(\langle l_i, v_i + t \rangle)$) [4],

[3] The constraint $0 \leq y < 1$ means that the tester can test whether clock y is in $[0, 1)$, but she cannot/need not read the exact value of y. In practice, this predicate can be implemented as a countdown timer whose timeout can be externally observed by the tester.

[4] If c_i is *delay*, because by definition for any state $\langle l, v \rangle$ we have $delay \notin enable(\langle l, v \rangle)$, then $\sigma_i \in \mathsf{R}_{>0}$ can be any time period that does

- if σ_i and σ_{i+1} are both in $\mathsf{R}_{>0}$ then $\xi_P(\langle l_i, v_i\rangle) \neq \xi_P(\langle l_{i+1}, v_{i+1}\rangle)$.

If a play ρ is a finite sequence, then it is called a *prefix*. The set of all prefixes of a TGA is denoted as Pref.

A *strategy* for a TGA is a function λ : Pref $\rightarrow (Act_c \cup \{delay\})$. Thus it is a trace-based (history-based) strategy.

If a player plays the game always according to what a strategy λ suggests him to do, the resulting prefix is called a λ-*supervised prefix*.

An *observation history* with P is a function Obs_P : Pref $\rightarrow O_P^*$, which maps a prefix ρ to the chronological sequence $\mathsf{Obs}_P(\rho)$ of non-stuttering observations along ρ.

Let λ be a strategy for the controller. Along any prefix, if the observation does not change at the next state implies that λ does not suggest a new move at that state, then λ is called an *observation-based stuttering-invariant* (OBSI) strategy.

Given a reachability property φ = control: A$\langle\rangle$ ψ stating whether ψ can always be eventually satisfied, an OBSI strategy λ for the controller is *winning* w.r.t. φ if there exists a λ-supervised prefix which has a trailing observation that satisfied φ.

Given a network of bounded TGA models of the controller and the plant, a set of observable predicates, and a winning objective in terms of e.g. reachability or safety property, we check whether the property can be satisfied by the models using knowledge-based subset construction and on-the-fly partially observable reachability (OTFPOR) computation [8], which is based on a mixture of forward search and backwards propagation timed game solving algorithm [7]. If the outcome is positive, a winning strategy for the controller will be extracted from the explored paths.

2.3 An example strategy

For a reachability property φ, a winning OBSI strategy can be represented as a directed acyclic graph, which has an initial node corresponding to the initial state and a number of leaf nodes corresponding to the states that satisfy φ.

For the light controller example and the reachability property "control: A$\langle\rangle$ Bright", Fig. 3 shows a winning OBSI strategy λ_R that is generated by our partially observable timed game solver. Each node in Fig. 3 corresponds to an observation history, and each of its outgoing edges correspond to a macro step. The states and observation for each strategy node are given in Table 1, where for simplicity instead of listing a complete observation for each node in the "Observations" column, we list only those observable predicates that evaluate to true under that observation.

In Fig. 1 we consider the following prefix:

not incur new observation; if $c_i \in Act_c$, then it means that time period $\sigma_i \in \mathsf{R}_{>0}$ can elapse only when during this period c_i is not enabled. In other words, if the controller prefers *delay*, or her preferred input is not enabled, then the plant can delay.

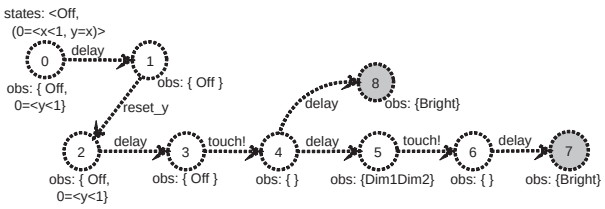

Figure 3. A winning strategy λ_R for reachability property control: A$\langle\rangle$ Bright**.**

Table 1. The states and observations in λ_R.

Node	TA symbolic states	Observations
0	$\{\langle$Off, $(0 \le x < 1, y - x = 0)\rangle\}$	$\{$Off, $0 \le y < 1\}$
1	$\{\langle$Off, $(x = 1, y - x = 0)\rangle\}$	{Off}
2	$\{\langle$Off, $(1 \le x < 2, y - x = -1)\rangle\}$	$\{$Off, $0 \le y < 1\}$
3	$\{\langle$Off, $(x = 2, y - x = -1)\rangle\}$	{Off}
4	$\{\langle$L1, $(0 \le x \le 2, y - x = 1)\rangle\}$	{}
5	$\{\langle$Dim1, $(0 \le x < 3, y - x = 1)\rangle$, $\langle$Dim2, $(0 \le x < 3, 2 \le y - x < 4)\rangle\}$	{Dim1Dim2}
6	$\{\langle$L2, $(0 \le x \le 2, 2 \le y - x < 5)\rangle\}$	{ }
7	$\{\langle$Bright, $(0 \le x \le 2, 2 \le y - x < 5)\rangle\}$	{Bright}
8	$\{\langle$Bright, $(0 \le x \le 2, y - x = 1)\rangle\}$	{Bright}

Observable predicates: Off: ({Off}, true), $0 \le y < 1$: $(L, 0 \le y < 1)$, Dim1Dim2: ({Dim1,Dim2}, true), Bright: ({Bright}, true).

$\langle$Off, $(x = 0, y = 0)\rangle \cdot$ *delay* $\cdot\, 1 \cdot \langle$Off, $(x = 1, y = 1)\rangle \cdot$ *resety* $\cdot$ *resety* $\cdot$
$\langle$Off, $(x = 1, y = 0)\rangle \cdot$ *delay* $\cdot\, 1 \cdot \langle$Off, $(x = 2, y = 1)\rangle \cdot$ *touch* $\cdot$ *touch* $\cdot$
$\langle$L1, $(x = 0, y = 1)\rangle \cdot$ *delay* $\cdot\, 0 \cdot \langle$Dim1, $(x = 0, y = 1)\rangle \cdot$ *touch* $\cdot\, 1 \cdot$
$\langle$Dim1, $(x = 1, y = 2)\rangle \cdot$ *touch* $\cdot$ *touch* $\cdot \langle$Dim2, $(x = 0, y = 2)\rangle \cdot$ *touch* $\cdot 1 \cdot$
$\langle$Dim2, $(x = 1, y = 3)\rangle \cdot$ *touch* $\cdot$ *touch* $\cdot \langle$L2, $(x = 0, y = 3)\rangle$.

The observation history of this prefix is:
$\{$Off, $0 \le y < 1\} \cdot \{$Off$\} \cdot \{$Off, $0 \le y < 1\} \cdot \{$Off$\} \cdot \{\} \cdot \{$Dim1Dim2$\} \cdot \{\}$.

The above history corresponds to the trace 0-1-2-3-4-5-6 in Fig. 3. To illustrate the idea of OBSI strategies, let us consider the macro step from node #5 to node #6 in Fig. 3, which has been zoomed out and instantiated from a concrete state $\langle$Dim1, $(x = 0, y - x = 1)\rangle$ in Fig. 4. This macro step consists of 4 micro steps, during all of which the controller is advised to offer a *touch*!. Unfortunately, in states $\langle$Dim1, $(x = 0, y - x = 1)\rangle$ ("5a") and $\langle$Dim2, $(x = 0, y - x = 2)\rangle$ ("5c") the touch transitions are not enabled. Thus the controller fails to carry out his move and instead the uncontrollable plant moves (here the 1 time unit delays) take place.

Note that there may exist non-deterministic transitions in a strategy, i.e., for a given observation history and for the same controller move, there might exist more than one possible next observation. For instance, in Fig. 3, if the user sticks to the *delay* move at node #4, then there may be a new observation of either that in node #5 or that in node #8. This non-determinism is due to the reaction uncertainties in the TGA models. Considering this non-determinism, we represent a strategy λ using a directed graph (N, E) where

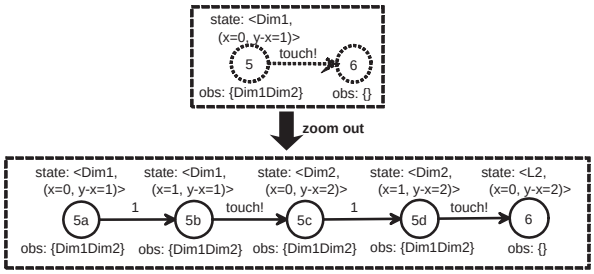

Figure 4. Macro step vs. micro step in λ_R.

N is the set of nodes, and $E = N \times (Act_c \cup \{delay\}) \times N$ is the set of edges. An initial node n_0 corresponds to the initial state of the TGA. Each edge $e = (n, mov, n') \in E$ denotes that if at node n the user sticks to mov, then there will be a new observation at node n'.

3 Timed conformance testing

3.1 Test setup

If a winning OBSI strategy for the controller can be synthesized, then it implies that the parallel composition of the plant model and the strategy-constrained (or -refined) controller model satisfies the property in question. Now our problem is how to test whether the property (referred to as *test purpose* in this context) is also satisfied by the various implementations IMP of the plant model.

In this paper we propose to use winning OBSI strategies as test cases on the plant implementations. In case a positive test verdict comes out, the IMP is said to satisfy the test purpose; otherwise, it does not. Fig. 5 shows our testing framework, where on-the-fly partially observable reachability (OTFPOR) computation was discussed earlier, and test execution will be discussed in this section.

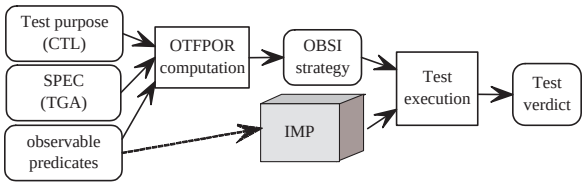

Figure 5. The testing framework.

Conformance testing demands a suitable implementation relation. Existing timed conformance relations, e.g., those built on top of trace equivalence [12, 20] or on top of observable behavior inclusion (e.g., ioco$_{DTA}$ [14], tioco [13, 15], tioco$_M$ [6], rtioco [17]), are not directly based on the partial observations that are made on the IMP. In this paper, we view the IMP as a "grey-box" in the sense that some of the IMP internals are observable but some others are not observable by the tester (controller). We define a notion of conformance in terms of partial observations. Fig. 6 is a schematic view of observation-based conformance testing of partially observable timed systems.

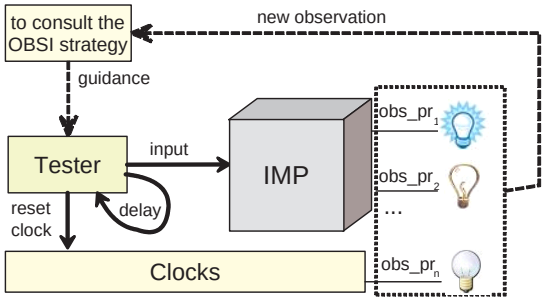

Figure 6. Observation-based conformance testing.

In Fig. 6, each observable predicate obs_pr$_i$ can be thought of as a "probe" into the IMP to detect whether the values of system variables and clocks are within some particular intervals, or a "location sensor" to report whether some particular locations have been arrived. These could be realistic assumptions, because we are usually only able to make limited precision measurements, and we usually have some guarding sensors which just monitor whether some valve values are reached and do not care about the exact values at all. In order to keep track of the timing information of the system precisely enough such that the clock constraints in transition guards can somehow be satisfied, the tester might need some private clocks, e.g. clock y in Fig. 1. The tester has three choices: to stick to a controllable input action, to reset the clocks, or to keep on delaying.

Given a deterministic and internal-action-free TA, we can prove that any externally observable timed I/O sequence uniquely corresponds to a history of TA state changes under full observability. The relaxation from full observability of TA internals to partial observability of them enables us to cope with a variety of applications that are previously based on stricter assumptions. To some extent this also helps to realize the potential of design for testability by allowing us to test a family of different IMP's of the same SPEC.

In Fig. 6, the tester assumes the role of the user (controller) in the timed game, and IMP assumes the plant role. In the beginning of every tester/IMP interaction session, the tester consults the OBSI strategy using the observation history obtained so far, and in return gets an instruction on what move she is supposed to make during this session: either to stick to a particular controllable input action, or to stick to *delay*. By "stick to" we mean that the tester can arbitrarily repeat the same move (if it is a controllable action

then it should be enabled) during the session until a new observation occurs, regardless whatever response the IMP makes. A new observation signifies the start of a next session of tester/IMP interactions. In Fig. 5, the tester makes test verdicts based on the observations, the OBSI strategy, and a conformance relation between the SPEC and the IMP.

3.2 Observation-based conformance

To decide whether the IMP is a correct implementation of the SPEC model, in this paper we define a partial observation-based conformance relation poco.

Similar to the definition of the input/output conformance relation ioco [21], the idea of poco is that after the tester plays an arbitrary number of sessions with the IMP using the moves that are suggested by the strategy and enabled in the SPEC model, each possible next new observation that the IMP could exhibit should be allowed by the SPEC.

Let $\rho = \langle l_0, v_0\rangle c_0\sigma_0\langle l_1, v_1\rangle c_1\sigma_1 \ldots \langle l_n, v_n\rangle$ be a prefix, P be a set of observable predicates, and let $\mu \in (Act_c \cup \{delay\})$ be the move that the controller sticks to during one interaction session. We define the set of possible new prefixes where a new observation has just occurred as

$$\rho\ \mathsf{After}_P\ \mu = \{\langle l_0, v_0\rangle c_0\sigma_0\langle l_1, v_1\rangle c_1\sigma_1 \ldots \langle l_n, v_n\rangle c_n\sigma_n \langle l_{n+1}, v_{n+1}\rangle c_{n+1}\sigma_{n+1} \ldots \langle l_m, v_m\rangle\},$$

where for all $n \le i \le m-1$, we have $c_i = \mu$, $\langle l_i, v_i\rangle \xrightarrow{\sigma_i} \langle l_{i+1}, v_{i+1}\rangle$, $\xi_P(\langle l_i, v_i\rangle) = \xi_P(\langle l_n, v_n\rangle)$, $\xi_P(\langle l_m, v_m\rangle) \neq \xi_P(\langle l_{m-1}, v_{m-1}\rangle)$, and the c's and σ's satisfy the constraints in Definition 4.

In the above definition μ can be extended from a single move to a string of moves in $(Act_c \cup \{delay\})^*$. Specifically, when μ is ε (i.e., empty string), we define $\rho\ \mathsf{After}_P\ \varepsilon = \{\rho\}$.

For instance, in the example in Fig. 1, we have: $\langle \mathsf{Off}, x = 0, y = 0\rangle\ \mathsf{After}_P\ delay = \{$
$\langle \mathsf{Off}, x = 0, y = 0\rangle \cdot delay \cdot 1 \cdot \langle \mathsf{Off}, x = 1, y = 1\rangle,$
$\langle \mathsf{Off}, x = 0, y = 0\rangle \cdot delay \cdot 0.5 \cdot \langle \mathsf{Off}, x = 0.5, y = 0.5\rangle \cdot delay \cdot 0.5 \cdot \langle \mathsf{Off}, x = 1, y = 1\rangle,\ \ldots\}$.

The definition of observation history in Section 2.2 can be extended from a single prefix to a set of prefixes, i.e., $\mathsf{Obs}_P : 2^{\mathsf{Pref}} \to 2^{O_P^*}$. Let $\mathsf{pref\_set} = \{\rho_1, \rho_2, \ldots, \rho_n\}$, we have

$$\mathsf{Obs}_P(\mathsf{pref\_set}) = \bigcup_{1 \le i \le n} \{\mathsf{Obs}_P(\rho_i)\}.$$

For instance, let the initial state in Fig. 1 be s_0, then we have $\mathsf{Obs}_P(s_0\ \mathsf{After}_P\ delay) = \{\{\mathsf{Off}, 0 \le y < 1\} \cdot \{\mathsf{Off}\}\}$, and $\mathsf{Obs}_P(s_0\ \mathsf{After}_P\ delay\ resety\ delay\ touch!\ delay) = \{\{\mathsf{Off}, 0 \le y < 1\} \cdot \{\mathsf{Off}\} \cdot \{\mathsf{Off}, 0 \le y < 1\} \cdot \{\mathsf{Off}\} \cdot \{\} \cdot \{\mathsf{Bright}\}, \{\mathsf{Off}, 0 \le y < 1\} \cdot \{\mathsf{Off}\} \cdot \{\mathsf{Off}, 0 \le y < 1\} \cdot \{\mathsf{Off}\} \cdot \{\} \cdot \{\mathsf{Dim1Dim2}\}\}$.

Definition 5 (**Partial Observation-based COnformance relation**, poco). Let ρ, η be two prefixes of the timed game execution, and P a set of observable predicates. The *partial observation-based conformance* relation poco_P between ρ and η is defined as:

$$\rho\ \mathsf{poco}_P\ \eta \text{ iff } \forall \mu \in (Act_c \cup \{delay\})^*.(\mathsf{Obs}_P(\rho\ \mathsf{After}_P\ \mu) \subseteq \mathsf{Obs}_P(\eta\ \mathsf{After}_P\ \mu)).$$

Let the TLTS of the SPEC model be $\mathcal{S}$, whose initial state is s_0 (note that s_0 is also a prefix), and assume that the behavior of the implementation IMP can be modeled by a TLTS $\mathcal{I}$, who can accept the same set of input actions and has the same set of observable predicates as $\mathcal{S}$, and assume the initial state of $\mathcal{I}$ to be i_0. If $i_0\ \mathsf{poco}_P\ s_0$, then we say $\mathcal{I}$ is a *correct implementation* of $\mathcal{S}$, denoted $\mathcal{I}\ \mathsf{poco}_P\ \mathcal{S}$.

While traditional conformance relations (such as tioco [13, 15], tioco_M [6], rtioco [17]) are based on sequences of I/O events, our poco is based on sequences of system observations. While the former relations accommodate partial observability in terms of internal events only, poco in addition considers the inaccuracies of measurements. Consequently, even if there is no internal action in the system, the system could still be partially observable. Thus we have a more expressive notion of partial observability.

3.3 Test execution algorithms

To execute a winning OBSI strategy on an IMP as a test case, the tester should stick to a specific move as suggested by the strategy. If the observation of the IMP does not change after a micro step, then the tester can offer the same input action to the IMP (if this input action is enabled in the SPEC) again, or delay again. Once the IMP observation changes, the tester checks whether this new observation is allowed by the SPEC model. If allowed, then a next session of tester/IMP interaction will start; otherwise, an error has been revealed. When computing the OBSI strategy, those sets of allowed next observations are included in the strategy, thus during test execution it is not necessary to consult the SPEC model.

Let $obsv(\mathsf{IMP})$ be an instantaneous observation (or a "snapshot") of the IMP, $\lambda = (N, E)$ be a winning OBSI strategy for a reachability test purpose, and $n_0 \in N$ be the initial strategy node. Let $n \in N$, we define

- the observation at strategy node n as $obs(\lambda, n)$,
- the strategy-suggested move at node n as $move(\lambda, n) = mov \in (Act_c \cup \{delay\})$ such that $\exists n' \in N.(n, mov, n') \in E$, and
- the set of strategy-allowed next observations following node n as $suc\_obs(\lambda, n) = \{obs(\lambda, n') | \exists mov \in (Act_c \cup \{delay\}).(n, mov, n') \in E\}$.

We use variable $node$ to track the current position in a strategy, imp_obs and imp_obs' to hold recent observations of the IMP, and mov to record the strategy-suggested move. If imp_obs satisfies the reachability test purpose, then we say imp_obs is a winning observation. For instance, the observations $\{\mathsf{Bright}\}$ and $\{\mathsf{Bright}, 0 \leq y < 1\}$ are both winning w.r.t. control: A$\langle\rangle$ Bright. Algorithm 3.1 describes the test execution towards a reachability test purpose φ using the IMP and a winning OBSI strategy λ for φ.

Algorithm 3.1 **TestExec_Reachability**(λ, IMP)

Input: winning strategy λ, system implementation IMP;
Output: test verdict pass or fail;
Initialization:
$node := n_0$;
$imp\_obs := obsv(\mathsf{IMP})$;
if $imp\_obs \neq obs(\lambda, node)$ then return(fail);
Main:

1: while (imp_obs is not winning) do
2: $imp\_obs' := imp\_obs$;
3: $mov := move(\lambda, node)$;
4: repeat
5: if $mov \in Act_c$ then offer mov to IMP;
6: $imp\_obs := obsv(\mathsf{IMP})$;
7: until $imp\_obs \neq imp\_obs'$;
8: if $imp\_obs \notin suc\_obs(\lambda, node)$ then return(fail);
9: else $node := n \in N$ such that $obs(\lambda, n) = imp\_obs$ and $(node, mov, n) \in E$; // the second $node$ takes the old value
10: endwhile
11: return(pass).

Having said that a winning OBSI strategy for a reachability test purpose is a directed acyclic graph ending with a number of observations that satisfy the test purpose. Clearly, the strategy provides only finite length guidance for the tester. This ensures that the while loop in Algorithm 3.1 eventually terminates.

In the repeat loop (lines 4-7), there are seemingly zenoness problems and racing problems between the tester and the SUT, i.e., sticking to an input action or clock resets without leading to a new observation might block the time, and sticking to *delay* might require the tester to make infinite frequent observations of the SUT. These, however, can both be avoided. In the former case, the generated strategy ensures that there will be no self-loop with any symbolic state, and in the latter case, we can implement the delaying by using the sleep and wake-up mechanisms.

The time complexity of Algorithm 3.1 depends on the length of the strategy, the shape of the strategy (i.e., the sizes of the sets suc_obs), and the lengths of the tester/IMP interaction sessions (i.e., how soon will there be a change of IMP observation after the provision of a test stimulus). Let p be the strategy length, $m = \max_{0 \leq i \leq p-1} \{|suc\_obs|\}$ be the maximal size of those suc_obs. Recall that the clocks in the TGA models are bounded by a maximal constant M, this means that the cumulative delay time between any two adjacent observations in the SPEC TLTS could also be bounded by a constant, say $w = k \cdot \mathsf{M}$. Then the worst-case time complexity of Algorithm 3.1 is $O(p \cdot (w + m))$.

3.4 Soundness and completeness

An ideal test suite or test case should exhibit both the soundness and the completeness (or "exhaustiveness") properties. A winning OBSI strategy executed as a test case is sound if there is no false positive (i.e, a detected error is really an error), and it is complete w.r.t. the given test purpose φ if there is no false negative (i.e., if there is an error related to φ, then there exists a test case to detect it).

Let the TLTS of the SPEC TGA be $\mathcal{S}$, whose initial state is s_0, and assume that the behavior of the implementation IMP can be modeled by a TLTS $\mathcal{I}$, which can accept the same set of input actions and has the same set of observable predicates as $\mathcal{S}$, and let the initial state of $\mathcal{I}$ be i_0.

Theorem 6 (**Soundness**). If there is a fail verdict in test execution, then $\mathcal{I}$ ~~poco~~ $\mathcal{S}$.

Let φ be a reachability test purpose that is satisfied by the SPEC model. Let $\mathcal{S}_\varphi$ be the behavior of $\mathcal{S}$ that are constrained by φ, i.e., the TLTS obtained by removing those φ-violating runs from $\mathcal{S}$.

Theorem 7 (**Partial completeness**). If $\mathcal{I}$ ~~poco~~ $\mathcal{S}_\varphi$, then there exists a winning OBSI strategy (test case) λ for φ such that a fail verdict will occur when executing λ with the IMP.

The proofs can be found in a detailed version paper [11].

4 Experimental results

Our prototype tool [8] reads a network of TGA, together with a set of observable predicates, and a reachability test purpose in CTL formula, and generates a winning OBSI strategy. This section reports on some strategy (test case) generation experiments. More details can be found in [11].

(1) An extended smart light controller

In the example of Fig. 1 and 2 the Light has only four brightness levels: Off, Dim1, Dim2, and Bright. Now we consider a finer granularity of the brightness levels, say, level 0 is Off, level $MaxLevel$ is Bright, and there are $(MaxLevel - 1)$ stairs of Dim between Off and Bright. Furthermore, we add clock variables tp and z to the system for finer grained modeling of the timing constraints. Then

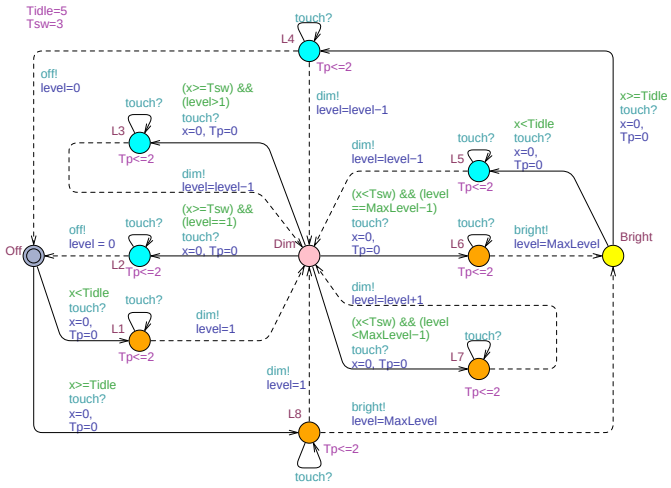

Figure 7. The smart light controller with multiple brightness levels.

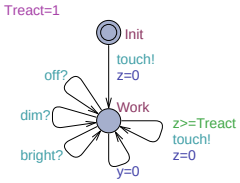

Figure 8. The user of the smart light.

we have extended TGA model for the smart light controller (Fig. 7) and the user (Fig. 8).

Compared to Fig. 1 and 2, the extended version has larger state space. There are 4 clocks (x, tp, z, and y) in the extended system, and an integer data variable $level \in [0, 10]$ is used to represent the current brightness level.

Our definition of observable predicates in Section 2.1 can actually be extended with a further dimension of data variable constraints, e.g., $3 \leq var \leq 6$. This extension has been implemented in the prototype tool. This enables us to better describe the observations.

In this example, the set of observable predicates is $P = \{$ ($\{$Off$\}$, true), ($\{$Bright$\}$, true), ($\{$L1, L6, L7, L8$\}$, true), ($\{$L2, L3, L4, L5$\}$, true), ($\{$Dim$\}$, $level = MaxLevel/2$), ($\{$Dim$\}$, $MaxLevel/2 + 1 \leq level \leq MaxLevel - 1$), ($\{Dim\}$, $1 \leq level \leq MaxLevel/2-1$), $(L, 0 \leq y <$ M$)\}$ where M is the upper bound for clock y.

The test purposes can be formulated as reachability properties (R_i) as follows:

R_1 control: A$\langle\rangle$ (Dim and $level = MaxLevel/2$)

Here, R_1 states that the tester wants to ensure that the Light always eventually reaches the "Medium Dim" state.

Table 2 presents the experimental results of test case generation for R_1. They include the time overheads, memory consumptions and strategy sizes (how many nodes inside the strategy). The leftmost column specifies which test purpose and what MaxLevel values (10, 20, or 30) we are using. Time granularity for the clock-related observable predicate, say $y \in [0, 1)$, is also considered. In Table 2, "/" means running out of memory.

Table 2. Results of test case generation.

	$y \in [0, 100)$	[0, 10)	[0, 2)	[0, 1)	[0, 0.5)	[0, 0.1)	[0, 0.01)
	⟨**time** (s), **memory** (KB), **strategy size** (# of nodes) ⟩						
R_1 10	0.67	0.66	0.87	1.26	1.44	1.38	1.39
	3828	3988	3924	4096	6220	6268	6272
	84	84	138	199	219	220	220
20	2.30	3.55	3.80	8.26	14.30	18.98	18.86
	6380	7192	7456	11964	15600	17308	17404
	334	468	504	952	1500	1645	1645
30	5.53	9.29	7.83	20.67	55.25	140.29	137.19
	7824	11468	11520	15556	30068	46992	46736
	709	996	840	1723	3514	5310	5270

Experiment platform: 2×2.00GHz CPU, 1024MB RAM; Ubuntu 8.04, Ruby 1.8.6, Ruby-BDD Binding 0.2, Uppaal DBM Library 2.0.6.

Table 3. Full vs. partial observation-based test case generation.

	full observation			partial observation		
	time (s)	**memory** (MB)	**strategy** (# of nodes)	**time** (s)	**memory** (MB)	**strategy** (# of nodes)
$(4, 4)$	489.05	2735	197136	18.33	168	56
$(5, 5)$	(out of memory)			137.34	761	81

Experiment platform: Sun Fire X4100, 2×2.4GHz CPU, 4096MB RAM; Suse Linux Enterprise Desktop 10 (64bit), Uppaal TIGA 0.13.

The results in Table 2 indicate that winning OBSI strategy generation for non-trivial TGA models are feasible in terms of both time overhead and memory consumption on an ordinary PC. We also find out that, in general, a coarser time granularity might lead to uncontrollability of the problem, whereas a finer granularity demands more resources to generate the test cases.

(2) A leader election protocol

We also carried out an experiment on test case generation for a leader election protocol [16] for mobile ad hoc networks. The TGA models of this protocol have n nodes, and n buffer cells which model the capacity of the communication channels. Each of the nodes and buffers needs one clock. Moreover, there are a global clock for counting the desired election time and a clock specifically for resetting. So there are altogether $2n + 2$ clocks. In addition, there are a number of global/local data variables to store and compare the message information. We defined a collection of observable predicates.

We did comparative studies of test generations experiments based on fully observable [10] and partially observ-

able models of this example. For the same test purpose, Table 3 shows the results for (4 nodes, 4 buffers) and (5 nodes, 5 buffers), respectively.

A remarkable finding is that the partial observation-based approach generates much smaller (shorter) test cases. This is in accordance with the rationale of our approach, because it in general makes a much coarser partitioning of the model statespace than the full observation-based one.

5 Conclusions and future work

We discuss the problem of model-based conformance testing of partially observable timed systems. The systems are modeled as a network of time game automata, the partially observable portions are defined using a set of predicates on the system states, and the test purposes are formulated as CTL formulas stating reachability properties. We use a previously-developed partially observable timed game solver to generate winning OBSI strategies for the test purposes, which in this paper are used as test cases for real-time conformance testing. We present a test framework, define a conformance relation poco, develop test execution algorithms, and prove the soundness and completeness properties. Experiments on test generations for some non-trivial examples show that this method yields encouraging results.

This method has some necessary ingredients to be used in software product line engineering. By modeling the variabilities of the product lines using the reaction uncertainties of our TGA models, we can generate test cases for testing a family of similar but different software products against their high-level requirements.

Compared with full observation-based conformance testing, the approach in this paper requires weaker assumptions and seems to generate smaller (shorter) test cases. These suggest that partial observation-based monitoring and testing has better prospects of being practically useful and being industrially adopted.

Future work includes: (1) More and larger case studies on test generation. This might involve improving the game solver towards better performance, better scalability, and further optimized strategies; (2) To provide guidelines for creating a sufficient/correct/consistent set of observable predicates; (3) Conformance testing for safety test purposes, whose corresponding OBSI strategies may provide infinite guidance to the tester; (4) Qualitative analysis and quantitative evaluation of the influences of the degrees of controllability and observability of the system on test generation.

Acknowledgements We thank the anonymous referees and Marius Mikučionis for their helpful comments. The research leading to these results has received funding from the EuropeanCommunity's Seventh Framework Programme under grant agreement no. 214755.

References

[1] R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *Proc. STOC'95*, pages 363–372. ACM Press, 1995.

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

[3] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *Proc. CAV'07*, LNCS 4590, pages 121–125, 2007.

[4] A. Blass, Y. Gurevich, L. Nachmanson, and M. Veanes. Play to test. In *Proc. FATES'05*, pages 32–46. Springer, 2005.

[5] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In *Proc. CAV'03*, 2003.

[6] L. B. Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In *Proc. FATES'04*, 2004.

[7] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *Proc. CONCUR'05*, 2005.

[8] F. Cassez, A. David, K. G. Larsen, D. Lime, and J.-F. Raskin. Timed control with observation based and stuttering invariant strategies. In *Proc. ATVA'07*, 2007.

[9] A. David, K. G. Larsen, S. Li, and B. Nielsen. Cooperative testing of timed systems. In *Proc. MBT'08*, 2008.

[10] A. David, K. G. Larsen, S. Li, and B. Nielsen. A game-theoretic approach to real-time system testing. In *Proc. DATE'08*, 2008.

[11] A. David, K. G. Larsen, S. Li, and B. Nielsen. Timed testing under partial observability. Technical report, http://www.cs.aau.dk/∼li/papers/icst09tr.pdf.

[12] A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi. Timed test cases generation based on state characterization technique. In *Proc. RTSS'98*, pages 220–229, 1998.

[13] A. Hessel, K. G. Larsen, B. Nielsen, P. Pettersson, and A. Skou. Time-optimal realtime test case generation using uppaal. In *Proc. FATES'03*, 2003.

[14] A. Khoumsi, T. Jéron, and H. Marchand. Test cases generation for nondeterministic real-time systems. In *Proc. FATES'03*, LNCS 2931, pages 131–146. Springer, 2003.

[15] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *Proc. SPIN'04*, 2004.

[16] L. Lamport. Real-time model checking is really simple. In *CHARME 2005*, pages 162–175, 2005.

[17] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using uppaal. In *Proc. FATES'04*, 2004.

[18] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proc. STACS'95*.

[19] B. Nielsen and A. Skou. Automated test generation from timed automata. *STTT*, 5(1):59–77, 2003.

[20] J. Springintveld, F. Vaandrager, and P. R. D'Argenio. Testing timed automata. *TCS*, 254(1-2):225–257, 2001.

[21] J. Tretmans. Testing concurrent systems: a formal approach. In *Proc. CONCUR'99*, LNCS 1664, pages 46–65, 1999.

[22] M. Yannakakis. Testing, optimizaton, and games. In *Proc. ICALP'04*, LNCS 3142, pages 28–45. Springer, 2004.