A Game-Theoretic Approach to Real-Time System Testing

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 presents a game-theoretic approach to the testing of uncontrollable real-time systems. By modelling the systems with Timed I/O Game Automata and specifying the test purposes as Timed CTL formulas, we employ a recently developed timed game solver UPPAAL-TIGA to synthesize winning strategies, and then use these strategies to conduct conformance testing of the systems. The testing process is proved to be sound and complete with respect to the given test purposes. Case study and preliminary experimental results indicate that this is a viable approach to real-time system testing.

1. Introduction

Model-based conformance testing of real-time systems has attracted increasing research interests in recent years. A large proportion of these work employ Timed Automata (TA) or its variants to model the systems. Among them some make the assumptions that the system TA model is output-urgent and has isolated outputs [14][7]. "Output urgency" means that if the system can produce an output, then it should produce that output immediately. "Isolated output" means that at any moment in time if the system can produce an output, then it cannot accept inputs and cannot produce a different output. These assumptions on TA contribute to the testability property [14]. However, in many cases they are unnecessarily strong.

In this paper we aim to relax these two assumptions and present a test method for *uncontrollable* real-time systems, i.e., systems with uncontrollable outputs and timing uncertainty of outputs. By "uncontrollable outputs" we mean that at a certain time point, it is the system under test that determines whether or which one of the several possible outputs will occur. By "timing uncertainty of outputs" we mean that the system under test can produce an output during a certain time interval rather than only at a fixed time point. The benefits of allowing uncontrollable outputs and timing uncertainty of outputs in the system models include:

- It allows the implementors some freedom;
- The tester is usually concerned only with the high-level requirements rather than the implementation details;
- Modelling with uncontrollable outputs and timing uncertainty of outputs is more succinct and natural.

Systems with uncontrollable outputs and timing uncertainty of outputs may be modelled by Timed Game Automata (TGA), which is a variant of TA with their actions partitioned into controllable ones and uncontrollable ones. A play of the timed game between the plant (modelling the system) and the controller (modelling the environment) is a run of the TGA towards a specified test purpose, say, "location `IUT.Bright` can always be eventually reached". We have already implemented a timed game solver UPPAAL-TIGA, which can check whether a specified Timed CTL test purpose can be satisfied by a TGA, and if so, it can synthesize a winning strategy. Since a winning strategy is a step-by-step guidance towards the goal states which satisfy the test purpose, it can be viewed as a test case. This opens up the possibility of generating tests for uncontrollable real-time systems.

Related work. There are much work on model-based conformance black-box testing of real-time systems based on TA or Timed Transition Systems [14][7][11][4] [9][8]. For the sake of testability, some of them assume that the TA are controllable in the sense that it should be possible for an environment to drive a TA through all of its transitions [14]. This in turn requires that the TA have output-urgency and isolated outputs. In this paper, both of these two requirements are cancelled.

Testing as a game problem for untimed systems has been studied in [15]. Dense-time control problem based on timed game automaton has been investigated and solved in [13]. As part of the UPPAAL toolbox, UPPAAL-TIGA [2][3] can synthesize winning strategies for TGA models and user-specified test purposes using a real on-the-fly algorithm

[5][12]. Thanks to a complete re-implementation of the first prototype [5], there has been a dramatic performance improvement over the past two years.

2. Test Setup

2.1. The timed control problem

In a timed control problem, the control program (or "controller") actively offers inputs to and passively observes outputs from the plant at appropriate time (or time periods), as shown in Fig. 1. For a given control objective we can possibly synthesize a control strategy, guided by which the control program ensures that the plant will be operating in a desired manner and thus fulfilling the control objective.

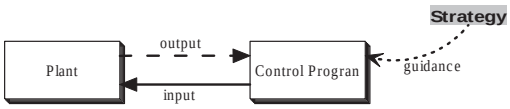

Figure 1. The timed control problem.

2.2. Timed I/O Game Automaton

Let X be a finite set of real-valued clocks, then $\mathcal{C}(X)$ is the set of constraints generated by the grammar:

$$\varphi ::= x \sim k \mid x - y \sim k \mid \varphi \wedge \varphi$$

where $k \in \mathbb{Z}$, $x, y \in X$ and $\sim \in \{<, \leq, =, \geq, >\}$.

Definition 1 (**Timed Automaton** [1]). A *timed automaton* (TA) is a tuple $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ where L is a finite set of locations, $l_0 \in L$ is the initial location, Act is the set of actions, X is a finite set of real-valued clocks, $E \subseteq L \times \mathcal{C}(X) \times Act \times 2^X \times L$ is a finite set of transitions, $Inv : L \to \mathcal{C}(X)$ associates to each location its invariant.

To characterize the uncontrollability of some actions, we adopt the notion of Timed Game Automaton [13].

Definition 2 (**Timed Game Automaton**). A *timed game automaton* (TGA) is a timed automaton with its set of actions Act partitioned into controllable ones (Act_c) and uncontrollable ones (Act_u).

In this paper we further refine the above definition by assuming all output actions Act_{out} to be uncontrollable and all input actions Act_{in} to be controllable.

Definition 3 (**Timed I/O Game Automaton**). A *timed I/O game automaton* (TIOGA) is a timed game automaton with its set of actions Act partitioned into input actions Act_{in} and output actions Act_{out} such that $Act_{in} = Act_c$ and $Act_{out} = Act_u$.

In a TIOGA, the controllable actions model the inputs from the controller (or the tester in the testing architecture) to the plant, and the uncontrollable actions model the outputs from the plant to the controller (or the tester). A *run* of the TIOGA involves a sequence of tester-chosen stimuli and plant-produced reactions. Therefore it can be viewed as a timed I/O game where the tester acts as a player and the plant acts as the opponent. Since sometimes the opponent may choose not to produce any output by just staying quiescent, the game is not a strictly alternating sequence of inputs and outputs.

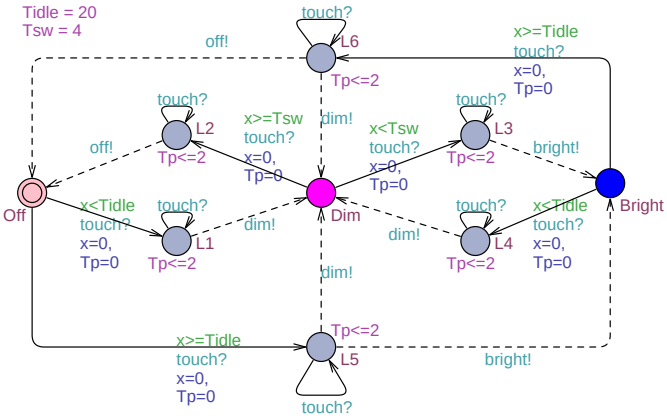

Figure 2. TIOGA of the light $\mathcal{S}$.

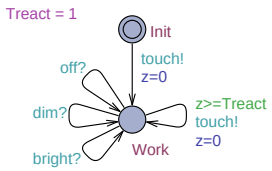

Figure 3. TA of the user $\mathcal{E}$.

This paper uses the simple Smart Light problem [7] as a running example. Fig. 2 is a TIOGA of the light (the "plant"), where solid lines represent transitions of controllable actions and dotted lines represent transitions of uncontrollable actions. Fig. 3 is the TA of the user or the "environment" of the light (the "controller"). The user interacts with the light by touching a touch-sensitive pad. In Fig. 2, there are three brightness levels for the light: `Off`, `Dim` or `Bright`. The light is initially in location `Off`. Assume that our goal is to reach location `Bright`. If the light has been in location `Off` for a long time ("$x \geq Tidle$" in Fig. 2), then it is supposed to reactivate upon a *touch*? and go to location `L5`, and then either to produce output *bright*! and go directly to location `Bright` in 2 time units, or to produce output *dim*! and go to location `Dim` in 2 time units, or even not to produce any output and remain in `L5` dur-

ing that period. The user does not know whether or which output will be produced. This is the so-called *output uncontrollability*. If the output is ever produced, the user cannot anticipate the exact time of output. This is the so-called *timing uncertainty of outputs*.

We use Timed I/O Transition System (TIOTS) as the underlying semantics model of TIOGA and TA.

Definition 4 (**Timed I/O Transition System**). A *timed I/O transition system* (TIOTS) is a tuple $(S, s_0, Act_{in}, Act_{out}, \rightarrow)$, where S is a set of states, $s_0 \in S$ is the initial state, and $\rightarrow \in S \times (Act_{in} \cup Act_{out} \cup \mathbb{R}_{\geq 0}) \times S$ is a transition relation satisfying the following sanity constraints:

- *time determinism*: $(s \xrightarrow{d} s') \wedge (s \xrightarrow{d} s'') \Rightarrow (s' = s'')$,
- *time additivity*: $(s \xrightarrow{d_1} s') \wedge (s' \xrightarrow{d_2} s'') \Rightarrow (s \xrightarrow{d_1+d_2} s'')$,

where $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers, $s, s', s'' \in S$, and $d, d_1, d_2 \in \mathbb{R}_{\geq 0}$.

Let $s \in S$ and $\alpha \in (Act \cup \mathbb{R}_{\geq 0})$. We write $s \xrightarrow{\alpha}$ if $\exists s' \in S. s \xrightarrow{\alpha} s'$. This notation can be extended to strings of observable actions and time delays as usual.

We define the following characteristics of TIOTS:

- A TIOTS has *isolated output* if $\forall s \in S. \forall \alpha \in Act_{out}. \forall \beta \in Act. (((s \xrightarrow{\alpha}) \wedge (s \xrightarrow{\beta})) \Rightarrow (\alpha = \beta))$,
- A TIOTS is *output-urgent* if $\forall s \in S. \forall \alpha \in Act_{out}. (s \xrightarrow{\alpha}) \Rightarrow \forall d \in \mathbb{R}_{>0}. (s \not\xrightarrow{d})$.

The semantics of a TA or a TIOGA (L, l_0, Act, X, E, Inv) is defined as a TIOTS $(S, s_0, Act_{in}, Act_{out}, \rightarrow)$, where $S \subseteq L \times \mathbb{R}^X$ is the set of semantic states of location and clock vector, $s_0 = (l_0, \overline{0})$ is the initial state, and $\rightarrow \subseteq S \times (Act_{in} \cup Act_{out} \cup \mathbb{R}_{\geq 0}) \times S$ satisfies both the sanity constraints and the following constraints:

- time transition: $(l, u) \xrightarrow{d} (l, u + d)$ if $\forall d' \in [0, d]. ((u + d') \models Inv(l))$,
- action transition: $(l, u) \xrightarrow{a} (l', u')$ if $\exists e = (l, a, g, r, l') \in E. ((u \models g) \wedge (u' = [r \rightarrow 0]u) \wedge (u' \models Inv(l')))$.

A run of the TIOGA is characterized by a timed trace. An *observable timed trace* $\sigma \in (Act \cup \mathbb{R}_{\geq 0})^*$ is of the form $\sigma = d_1 a_1 d_2 a_2 \ldots a_k d_{k+1}$. We define the set of observable timed traces of state s as:

$$\mathsf{TTr}(s) = \{\sigma \in (Act \cup \mathbb{R}_{\geq 0})^* | s \xrightarrow{\sigma}\}.$$

For a state s and a timed trace σ, $(s \ \mathsf{After} \ \sigma)$ is the set of states that can be reached after σ:

$$s \ \mathsf{After} \ \sigma = \{s' | s \xrightarrow{\sigma} s'\}.$$

The set of (immediately) observable outputs or delays at state s is defined as:

$$\mathsf{Out}(s) = \{a \in (Act_{out} \cup \mathbb{R}_{\geq 0}) | s \xrightarrow{a}\}.$$

The above two definitions can both be extended to sets of states as usual.

A *run* of a TIOGA (L, l_0, Act, X, E, Inv) is a timed trace (i.e., a sequence of alternating time and action transitions) in its TIOTS $(S, s_0, Act_{in}, Act_{out}, \rightarrow)$. We use $\mathsf{Runs}(s, \mathcal{S})$ to denote the set of all runs of $\mathcal{S}$ that start from state $s \in S$. Specifically, $\mathsf{Runs}(\mathcal{S})$ denotes the set of all runs of $\mathcal{S}$ that start from s_0. If σ is a finite run, we use $last(\sigma)$ to denote the last state of σ.

In this paper, we assume the system models (the "plant") have the following characteristics:

- determinism,
- strong input-enabledness,
- non-output-urgency (thus timing uncertainty of outputs is allowed), and
- non-isolated-output (thus uncontrollable outputs are allowed).

A TIOGA which has uncontrollable outputs and timing uncertainty is called an uncontrollable TIOGA. Likewise, its corresponding TIOTS is called an uncontrollable TIOTS.

Moreover, we can define the parallel composition of several TIOTS's in the usual manner.

2.3. Timed I/O conformance relation

To decide whether the behavior of the implementation under test (IMP) conforms to that of the system specification (SPEC), we use the Timed Input-Output Conformance relation.

Definition 5 (**Timed Input-Output Conformance relation**, tioco [9]). Let $i, s \in S$ be two states of a TIOTS. The *timed input-output conformance relation* tioco between i and s is defined as:

$$i \ \mathsf{tioco} \ s \ \text{ iff } \ \forall \sigma \in \mathsf{TTr}(s). (\mathsf{Out}(i \ \mathsf{After} \ \sigma) \subseteq \mathsf{Out}(s \ \mathsf{After} \ \sigma)).$$

Assume that the behavior of the IMP can be modelled by a TIOTS $\mathcal{I}$. Assumed the initial state of $\mathcal{I}$ is i_0, and the initial semantic state of the specification TIOTS $\mathcal{S}$ is s_0. If i_0 tioco s_0, we say that $\mathcal{I}$ is a correct implementation of $\mathcal{S}$, denoted $\mathcal{I}$ tioco $\mathsf{TIOTS}(\mathcal{S})$.

Let $\mathcal{S}$ be input-enabled with states $i, s \in S$. The conformance relation tioco can also be characterized in terms of timed trace inclusion as follows:

$$i \ \mathsf{tioco} \ s \ \text{ iff } \ \mathsf{TTr}(i) \subseteq \mathsf{TTr}(s).$$

2.4. Test purpose

This paper aims to conduct targeted rather than comprehensive testing of whether an IMP conforms to a SPEC. This means that we should have in mind a test purpose. In

this paper, we use annotated Timed CTL formulas to specify test purposes, e.g., `control: A ⟨⟩ IUT.Bright`, where `control` means it is a test purpose for a timed game. The formula says that whatever uncontrollable outputs the IMP may produce according to the SPEC model, we can always be guided to offer inputs or to delay such that we can reach the goal location `IUT.Bright`.

2.5. Test hypotheses

For the purpose of proving the soundness and completeness properties of our test method, we assume that the system implementation IMP can be modelled by a TIOTS and it has the same sets of input actions Act_{in} and output actions Act_{out} as the SPEC. Furthermore, the IMP is assumed to be controllable, i.e., it has the characteristics of isolated outputs and output urgency. This is reasonable since IMP is usually more deterministic than SPEC.

3. Testing with winning strategies

3.1. The testing framework

The framework of testing with winning strategies is illustrated in Fig. 4. The inputs to UPPAAL-TIGA are the TIOGA model of the plant, the TA model of its environment, and the test purpose in a formula of an extended subset of the TCTL logic. The output from UPPAAL-TIGA is a winning strategy. With the SPEC models, the winning strategy and the black-box implementation IMP we can do testing and produce the test verdict pass or fail.

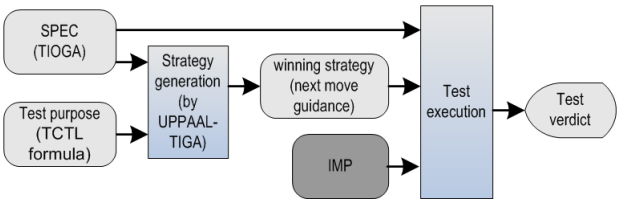

Figure 4. Testing with winning strategies.

3.2. Generating winning strategy

The key idea of our test method is to use a winning strategy as a test case. A test purpose for a reachability control problem is that given a TIOGA $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ and a set of goal states $K \subseteq L \times \mathbb{R}^X$ of its corresponding TIOTS, we should find a winning strategy f such that $\mathcal{S}$ supervised by f can reach some states in K. Obviously, the test purpose determines K.

We view the reachability control problem $(\mathcal{S}, K)$ as a game problem. A finite or infinite run of $\mathcal{S}$ $\sigma = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_{n+1}$ is *winning* if $\exists k \geq 0.(s_k \in K)$. The set of all winning runs in $\mathcal{S}$ starting from state s is denoted by $\mathsf{WinRuns}(s, \mathcal{S}, K)$. Winning runs in the underlying TIOTS are defined similarly.

A strategy f is a function that during the course of the timed game constantly gives information as to what the player should do in order to win the game [13]. At a given state of the run, the player can be guided either to do a particular controllable action (i.e., to offer an input to the plant), or to do nothing at this point in time and just wait (denoted by " λ ").

Definition 6 (**State-Based Strategy**). Let $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ be a TIOGA, and let $(S, s_0, Act_{in}, Act_{out}, \rightarrow)$ be the TIOTS of $\mathcal{S}$. A *state-based strategy* f over $\mathcal{S}$ is a partial function from S to $Act_c \cup \{\lambda\}$.

Definition 7 (**Supervised Run**). Let $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ be a TIOGA and f a state-based strategy over $\mathcal{S}$. Let s be a state in the TIOTS of $\mathcal{S}$. The *f-supervised runs* of $\mathcal{S}$ from s is a subset $\mathsf{SupRuns}(s, f) \subseteq \mathsf{Runs}(s, \mathcal{S})$ defined inductively as:

- $s \in \mathsf{SupRuns}(s, f)$,
- $\sigma' = (\sigma \xrightarrow{e} s') \in \mathsf{SupRuns}(s, f)$ if $\sigma \in \mathsf{SupRuns}(s, f)$, $\sigma' \in \mathsf{Runs}(s, \mathcal{S})$ and one of the following three conditions holds:
  - $e \in Act_u$,
  - $e \in Act_c$ and $e = f(last(\sigma))$,
  - $e \in \mathbb{R}_{\geq 0}$ and $\forall e' \in [0, e).\exists s'' \in S.((last(\sigma) \xrightarrow{e'} s'') \wedge (f(s'') = \lambda))$,
- $\sigma \in \mathsf{SupRuns}(s, f)$ if σ is an infinite run whose finite prefixes are all included in $\mathsf{SupRuns}(s, f)$.

For a reachability game with $K \subseteq L \times \mathbb{R}^X$, a *maximal run* σ is either an infinite run, or a finite run such that either $last(\sigma) \in K$, or $(last(\sigma) \notin K) \wedge ((last(\sigma) \xrightarrow{\alpha}) \Rightarrow (\alpha = 0))$. We denote the set of all maximal runs from state s as $\mathsf{MaxRuns}(s)$.

Let $\sigma = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_{n+1}$ be a run of TIOGA $\mathcal{S}$, and K be a set of goal states. If σ is a maximal run, then σ is *losing* if $\forall 0 \leq k \leq min\{index(last(\sigma)), \infty\}.(s_k \notin K)$.

Definition 8 (**Winning Strategy**). Let $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ be a TIOGA and f a state-based strategy over $\mathcal{S}$. Let s be a state in the TIOTS of $\mathcal{S}$. We say f is *winning* from state s if $\mathsf{MaxRuns}(s) \cap \mathsf{SupRuns}(s, f) \subseteq \mathsf{WinRuns}(s, \mathcal{S}, K)$. If f is winning from s_0, then f is called a *winning strategy* for $\mathcal{S}$.

A strategy being winning means that if the controller acts strictly according to what the strategy suggests, then what-

ever responses the plant might make, the controller will for sure be winning.

Fig. 5 shows a state-based winning strategy for the TIOGA in Fig. 2 and test purpose `control: A ⟨⟩ IUT.Bright`. It is automatically generated by UPPAAL-TIGA.

Note that there may exist more than one winning strategy for the same TIOGA and test purpose. We use $\mathsf{Strategy}(\mathcal{S}, \varphi)$ to denote the set of all winning strategies for TIOGA $\mathcal{S}$ and test purpose φ.

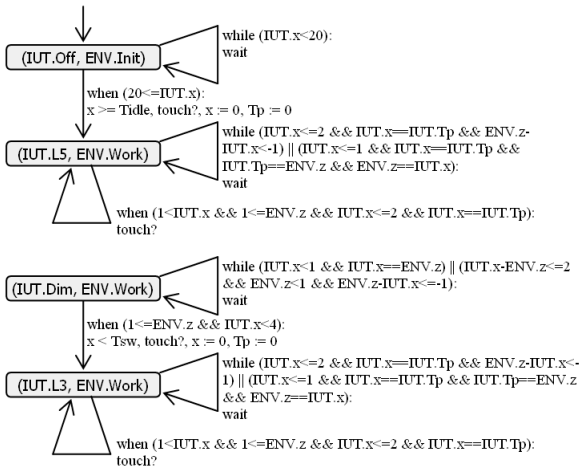

Figure 5. An example winning strategy.

3.3. Test execution

Definition 9 (**Test Execution**). Let $SPECS$ be the set of system specifications, F be the set of winning strategies, and $IMPS$ be the set of system implementations. A *test execution* is defined as a function:

$$T : SPECS \times F \times IMPS \rightarrow \{\mathsf{pass}, \mathsf{fail}\}.$$

The basic idea of test execution is to incrementally build a test run by constantly consulting the winning strategy and the SPEC model, as shown in Algorithm 3.1. If an occurred output is prohibited according to tioco, we report fail, otherwise after reaching a goal state we report pass.

3.4. Soundness and completeness

In conformance testing, the soundness property says that if there exists a failing test run, then the implementation does not conform to the specification.

Theorem 10 (**Soundness**). Let $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ be a TIOGA specification with $Act = Act_{in} \cup Act_{out}$, $\mathsf{TIOTS}(\mathcal{S})$ be its corresponding TIOTS, $\mathcal{I} = (I, i_0, Act_{in}, Act_{out}, \rightarrow)$ be a TIOTS implementation, and f be a winning strategy for $\mathcal{S}$, then $(\exists \sigma \in (\mathsf{SupRuns}(i_0, f) \cap \mathsf{MaxRuns}(i_0)).\ \sigma\ is\ losing) \Rightarrow (\mathcal{I}$ ~~tioco~~ $\mathsf{TIOTS}(\mathcal{S}))$.

Algorithm 3.1 **TestExec**($\mathcal{S}$, $\mathcal{I}$, K, f)

Input: TIOGA specification $\mathcal{S}$, system implementation $\mathcal{I}$, set of goal states K, and state-based winning strategy f;
Output: test verdict pass or fail, and test run σ;
Algorithm:

```
1:  σ := ⟨⟩; /* the test run is initially an empty trace */
2:  while (σ ∉ WinRuns(s0, S, K)) do  /*s0: init state*/
3:    case f(last(σ)) of
4:      "input i":
5:        send i to I;
6:        σ := σ·i;
7:      "delay d":
8:        if output o occurs at d' ≤ d then
9:          σ := σ·d';
10:         if o ∉ Out(s0 After σ) then
11:           return(fail);
12:         else
13:           σ := σ·o;
14:       else
15:         σ := σ·d;
16:   endcase
17: endwhile
18: return(pass).
```

In conformance testing, the completeness property says that if an implementation does not conform to a specification, then there exists a failing test run. In this paper, because we are conducting targeted testing with a test purpose, given a test purpose φ that is satisfied by the specification, if the implementation does not conform to the specification w.r.t. φ, we will be able to find a failing run. Hence the following theorem of partial completeness.

Theorem 11 (**Partial Completeness**). Let $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ be a TIOGA specification with $Act = Act_{in} \cup Act_{out}$, $\mathsf{TIOTS}(\mathcal{S})$ be its corresponding TIOTS, $\mathcal{I} = (I, i_0, Act_{in}, Act_{out}, \rightarrow)$ be a TIOTS implementation, and φ be a test purpose such that $\mathsf{TIOTS}(\mathcal{S}) \models \varphi$, then $(\mathcal{I}$ ~~tioco~~ $\mathsf{TIOTS}(\mathcal{S}))\ w.r.t.\ \varphi \Rightarrow \exists f \in \mathsf{Strategy}(\mathcal{S}, \varphi).\exists \sigma(\mathsf{SupRuns}(i_0, f) \cap \mathsf{MaxRuns}(i_0))$. ($\sigma$ is failing).

Proof details can be found in [6].

4. Case study

In this section we consider a simple Leader Election Protocol (LEP) [10] (more details in [6]), which is essentially a distributed consensus algorithm with timing constraints.

Table 1. Strategy generation for LEP protocol.

	Time (s)						**Memory** (MB)					
	n=3	4	5	6	7	8	n=3	4	5	6	7	8
TP1	0.03	0.14	0.7	3.1	11.1	33.5	0.1	4	9	28	85	242
TP2	0.81	2.13	8.4	67.1	452.0	/	11.2	33	88	462	2977	/
TP3	0.89	2.79	25.9	73.2	453.8	/	11.9	40	289	578	3015	/

The idea is to elect the node with the lowest address as the leader by using message passing.

We model the problem as two parts: one TIOGA for an arbitrary node as the plant, and two TA for its simulated chaotic environment including all the other nodes and a buffer with certain capacity as the controller. The TIOGA has uncontrollable actions in the sense that in the plant node a *timeout*! event can be produced at any point of a time frame after the node has been waiting for a certain period of time without receiving any "useful" messages.

We defined the following test purposes:

- TP1: `control: A ⟨⟩ (IUT.betterInfo == 1) and IUT.forward`
- TP2: `control: A ⟨⟩ forall (i: BufferId) (inUse[i] == 1)`
- TP3: `control: A ⟨⟩ forall (i: BufferId) (inUse[i] == 1) and IUT.idle`

All the above three test purposes are checked to be true by using UPPAAL-TIGA. We carried out the strategy generation experiments on an application server with dual-core 2.4GHz CPU, 4096MB RAM and Suse Linux Enterprise Desktop. Table 1 presents the performance results for these test purposes with different protocol parameter settings. The time and memory columns represent the time overheads and the memory consumptions for generating winning strategies for the test purposes, respectively. Each subcolumn corresponds to one parameter configuration, where n means that there are n nodes in the protocol, and there is a message buffer of size n, and the maximum distance between any two nodes is limited to $(n-1)$.

As can be seen from Table 1, winning strategy generation for the LEP protocol with up to 7 nodes takes less than 8 minutes and the memory consumption is not well beyond out expectation considering the complexity of the problem.

5. Conclusions and future work

We examine the problem of conformance black-box testing of uncontrollable real-time systems using a game-theoretic approach. We model the systems with timed I/O game automata and specify the test purposes with TCTL formulas. With the help of a recently developed timed game solver, we can do testing based on winning strategies. Experimental results of the Leader Election Protocol indicate that this approach is viable and computationally feasible. This opens up a new possibility for testing TA-modelled timed systems with timing uncertainty of outputs and uncontrollable outputs, which are previously thought of as somewhat under-specified.

Future work include: 1) generalizing state-based strategy to history-based strategy; 2) building a fully automated strategy-based testing environment, of which a big concern is efficient strategy representation; 3) evaluating strategy-based test effectiveness in terms of e.g. fault detecting capability; 4) if there does not exist a winning strategy, we hope to make a small "retreat" by doing cooperative testing; 5) strategy-based testing with partial observability.

References

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

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

[3] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. *UPPAAL TIGA User-manual*. Aalborg University, July 2007.

[4] L. B. Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In *FATES 2004*, pages 64–78, 2004.

[5] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005*, pages 66–80, 2005.

[6] A. David, K. G. Larsen, S. Li, and B. Nielsen. A game-theoretic approach to real-time system testing. Technical report, Aalborg University, November 2007.

[7] A. Hessel, K. G. Larsen, B. Nielsen, P. Pettersson, and A. Skou. Time-optimal realtime test case generation using uppaal. In *FATES 2003*, pages 114–130, 2003.

[8] C. Jard and T. Jéron. `TGV`: theory, principles and algorithms. *STTT*, 7(4):297–315, 2005.

[9] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *SPIN 2004*, pages 109–126, 2004.

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

[11] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using uppaal. In *FATES 2004*, pages 123–137, 2004.

[12] X. Liu and S. A. Smolka. Simple linear-time algorithms for minimal fixed points. In *ICALP 1998*, pages 53–66, 1998.

[13] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 1995*, pages 229–242, 1995.

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

[15] M. Yannakakis. Testing, optimizaton, and games. In *ICALP 2004*, pages 28–45, 2004.