Cooperative Testing of Uncontrollable Timed Systems

Alexandre David, Kim G. Larsen, Shuhao Li, and Brian Nielsen

Center for Embedded Software Systems (CISS)
Aalborg University
Selma Lagerlöfs Vej 300, DK-9220 Aalborg, Denmark
{adavid, kgl, li, bnielsen}@cs.aau.dk

Abstract. This paper deals with targeted testing of timed systems with uncontrollable behavior. The testing activity is viewed as a game between the tester and the system under test (SUT) towards a given test purpose. The SUT is modeled as Timed Game Automaton and the test purpose is specified in Timed CTL formula. We can employ a timed game solver UPPAAL-TIGA to check if the test purpose is ture w.r.t. the model, and if yes, to generate a winning strategy and use it for black-box conformance testing. Specifically, we show that in case the checking yields a negative result, we can still test the SUT against the test purpose as long as the SUT reacts to our moves in a cooperative style. We present an operational framework of cooperative winning strategy generation, test case derivation and execution. The test method is proved to be sound and complete. Preliminary experimental results indicate that this approach is applicable to non-trivial uncontrollable timed systems.

1 Introduction

In the field of model-based testing of real-time systems [5, 7, 9, 16, 11, 8, 13, 3, 12, 15], a considerable proportion of work [7, 9, 16, 11, 8, 13, 3, 12] employ timed automata or TTS to model the systems. Among them some make the assumptions that the system TA model is output-urgent and has isolated outputs [7, 16, 8]. "Output-urgent" means that if the system can produce an output, then it should be produced immediately. "Isolated output" means that anytime when the system can produce an output, it cannot accept inputs and cannot produce a different output. These two assumptions, together with the "determinism" assumption, contribute to the testability property [16] of timed automata by ensuring that given an input sequence fragment there is only one output emitted at a precise point in time, or lifted a little higher, by making it possible for an environment to "drive" a timed automaton through all of its transitions. However, in many cases the two assumptions are unnecessarily strong. For example in the simple Smart Light problem [8], the light model with output-urgency and isolated outputs (see Fig. 2) is too demanding in the sense that we should have one TA node exclusively for producing each output, and we should have strict timing of the output.

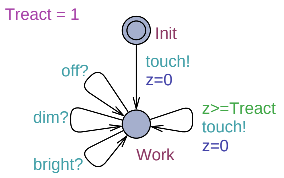


Fig. 1. The user.

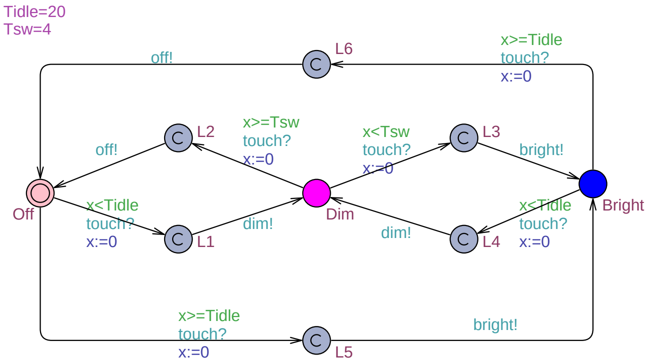


Fig. 2. Controllable TA of the light.

In this paper we aim to cancel the assumptions of isolated outputs and output-urgency, and present a test method for *uncontrollable* timed systems, i.e., systems with uncontrollable outputs and timing uncertainty of outputs. By “uncontrollable outputs” we mean that it is the system under test (SUT) rather than the tester that determines whether or which one of the several possible outputs will occur. By “timing uncertainty of outputs” we mean that the SUT can produce an output during a certain time interval rather than only at a fixed time point, or in other words, the timing of outputs is unpredictable by the tester. The benefits of permitting uncontrollable behavior in the system models include allowing the implementors some freedom, providing the tester with high-level or abstract requirements, and yielding more natural and succinct models.

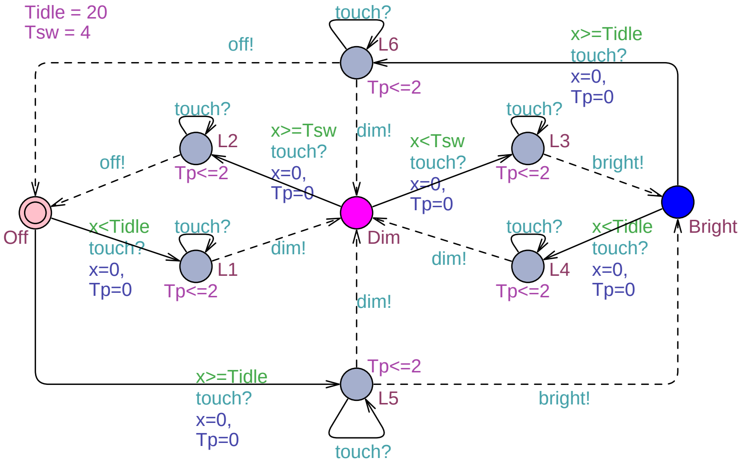


Fig. 3. TIOGA of an ideal light.

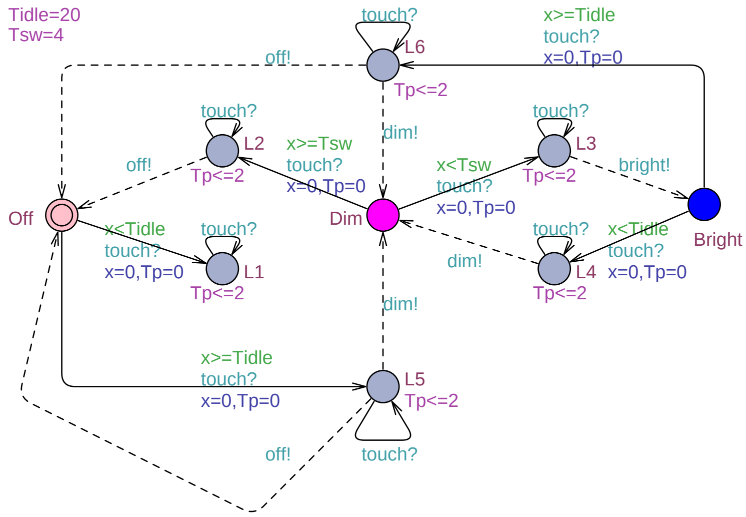


Fig. 4. TIOGA of a problematic light.

Systems with uncontrollable outputs and timing uncertainty of outputs may be modelled as timed game automata (TGA) [14], which is a variant of TA with their actions partitioned into controllable and uncontrollable ones. For example, Fig. 3 is a TGA of the light, where solid lines carry controllable actions (inputs) and dotted lines uncontrollable actions (outputs).

In a timed control problem, a control program (or “controller”, e.g. Fig. 1) actively offers inputs to and passively observes outputs from a plant that models the system in question (e.g., Fig. 3). A *run* of the system involves a sequence of controller-chosen stimuli and plant-produced reactions aiming to satisfy a given test purpose (e.g., “location `Bright` can always be eventually reached”). Therefore it can be viewed as a timed I/O game where the controller acts as a player and the plant acts as the opponent. 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 will thus fulfill the control objective.

The problem of dense-time controller synthesis has been solved using backwards fix-point computation [14]. As an improvement a truly on-the-fly algorithm [4] is proposed and has been implemented in the timed game solver UPPAAL-TIGA [2], which checks whether a user-specified test purpose can be satisfied by a TGA, and if so, it efficiently synthesizes a winning strategy for that test purpose. Specifically, in this paper we address the problem that in case an affirmative test purpose as above is checked to be false due to problematic TIOGA model (see Fig. 4) or too strong test purpose, we can make a “retreat” by relaxing the test purpose such that to some extent it requires cooperation from the plant, say, “location `Bright` can always be eventually reached *as long* *as* the system reacts to our moves in some desired manner". We use UPPAAL-TIGA to check whether this weakened test purpose can be satisfied, and if yes, to synthesize a *cooperative winning strategy.* Since a (cooperative) strategy is a step-by-step guidance towards the goal states of the model which fulfill the given test purpose, it can be viewed as a test and used for conformance testing [6].

From a game point of view, testing of untimed systems has been discussed in [18], but to our knowledge no such reported work for timed systems. Although strategy synthesis is inherently more expensive than some other approaches to timed testing, the idea and method proposed in this paper opens up the possibility of testing TA-modeled uncontrollable timed systems.

2 Test Setup

In this paper we endeavor to test whether a black-box system implementation IMP complies with its specification SPEC with respect to some given test purpose. As illustrated in Fig. 5, there are three steps in our testing framework: game strategy generation, test case generation, and test execution.

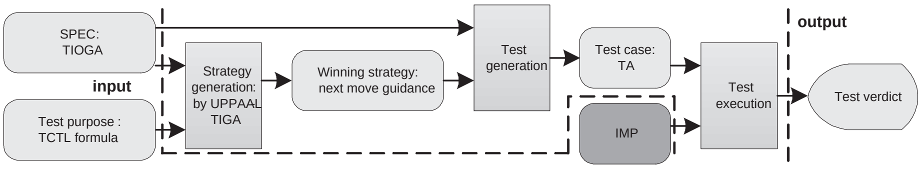


Fig. 5. The framework of strategy-based testing.

2.1 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 grammar $\varphi ::= x \sim k \mid x - y \sim k \mid \varphi \wedge \varphi$, where $k \in \mathbb{Z}$, $x, y \in X$ and $\sim \in \{<, \leq, =, \geq, >\}$.

A *timed automaton* (TA) [1] 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.

In timed game automaton [14], actions are partitioned into controllable ones and uncontrollable ones. Now we make a further assumption that all output actions Act_{out} are uncontrollable and all input actions Act_{in} are controllable.

A *timed I/O game automaton* (TIOGA) is a timed automaton with its set of actions Act partitioned into controllable actions Act_c and uncontrollable actions Act_u such that $Act_c = Act_{in}$ and $Act_u = Act_{out}$.

This paper uses the simple Smart Light problem [8] as an example. Fig. 1 is a TA of the user or the "environment" of the light (the "controller"). Fig. 4 is a "problematic" TIOGA of the light (the "plant"), where controllable actions (in solid lines) model the inputs from the controller to the plant, and uncontrollable actions (in dotted lines) model the outputs from the plant to the controller. The user interacts with the light by touching a touch-sensitive pad. In Fig. 4, there are three brightness levels for the light: `Off`, `Dim` and `Bright`. The light is initially in location `Off`. There are uncontrollable behavior in `L1`, `L2`, ..., `L6`.

The semantics of a TA or a TIOGA $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ is defined as a *timed I/O transition system* (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 the sanity constraints of time determinism and time additivity.

Let $s \in S$, and $\alpha \in (Act \cup \mathbb{R}_{\geq 0})$. If $\exists s' \in S . s \xrightarrow{\alpha} s'$, we write $s \xrightarrow{\alpha}$. Here α can be extended to strings of actions and time delays.

A *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 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, we define the set of states that can be reached after σ: $s \text{ After } \sigma = \{s' | s \xrightarrow{\sigma} s'\}$. If the set is a singleton, then we just denote it as the target state. The set of (immediately) observable outputs or delays at state s is defined as: $\mathsf{Out}(s) = \{a \in (Act_{out} \cup \mathbb{R}_{\geq 0}) | s \xrightarrow{a}\}$. These two definitions can be extended to sets of states as usual.

A *run* of a TIOGA $\mathcal{S}$ is a timed trace in its TIOTS. We use $\mathsf{Runs}(s, \mathcal{S})$ to denote the set of all runs of $\mathcal{S}$ that start from $s \in S$. Let $\mathsf{Runs}(\mathcal{S}) = \mathsf{Runs}(s_0, \mathcal{S})$. If σ is a finite run, then $last(\sigma)$ denotes the last semantic state of σ.

In this paper we only impose the "determinism" and "strong input-enabledness" restrictions on the TIOGA model of the plant. In particular we do not require the plant model to be output-urgent (thus allowing timing uncertainty of outputs), or have isolated outputs (thus allowing uncontrollable outputs). Such a TIOGA is called an *uncontrollable* TIOGA, and its corresponding TIOTS is called an *uncontrollable* TIOTS.

The parallel compositions of several TIOGA (TA) or several TIOTS's can be defined in the usual manner.

2.2 Timed Conformance Relation

Definition 1 (**Timed Input-Output Conformance relation**, tioco [12]). 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:

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

As test hypothesis we assume that the behavior of the IMP can be modelled by a TIOTS $\mathcal{I}$, which has the same sets of input actions Act_{in} and output actions Act_{out} as the specification TIOGA $\mathcal{S}$. Let the initial state of $\mathcal{I}$ be i_0, and the initial semantic state of $\mathcal{S}$ be s_0. If i_0 tioco s_0, we say that $\mathcal{I}$ is a correct implementation of the specification, denoted $\mathcal{I}$ tioco $\mathsf{TIOTS}(\mathcal{S})$. Furthermore, $\mathcal{I}$ is assumed to be deterministic and controllable.

We can also user other timed versions [11, 13, 15] of Tretmans' ioco relation [17].

2.3 Test Purpose

We aim to conduct targeted rather than comprehensive testing of whether an IMP conforms to a SPEC, thus we use a test purpose [10]. In this paper, we use annotated Timed CTL formulas to specify test purposes, e.g., control: A $\langle\rangle$ `Bright` means that whatever uncontrollable outputs the system may produce according to the SPEC model, we can always choose to trigger input transition or to delay such that the system is guaranteed to reach the goal location `Bright`. In the weakened case we write E $\langle\rangle$ control: A $\langle\rangle$ `Bright`, which means that we can always be guided to reach `Bright` *as long as* the system is willing to cooperate with us by producing outputs in some desired manner.

3 Cooperative Winning Strategy

A reachability control problem is that given a TIOGA $\mathcal{S}$ and a set of goal states K of its corresponding TIOTS, we should find a game strategy f such that $\mathcal{S}$ supervised by f can reach some states in K. If a state in K is reached, then the play of the game is said to be *winning*.

A strategy f is a function that during the course of a timed game constantly gives information as to what the player (the "controller") should do in order to win the game against the opponent (the "plant"). At a given state of the run, the player can be guided either to offer a particular input and bring it to a particular state, or to do nothing at this time point and just wait (denoted "λ").

When a test purpose φ is checked to be true, then there must exist a winning strategy for φ. A strategy being winning means that if the controller acts strictly according to what the strategy suggests, then whatever responses the plant might make, the behavior of the plant will satisfy the test purpose.

When a test purpose φ is checked to be false, then there does not exist a winning strategy. For example in the TIOGA of Fig. 4, there is no winning strategy for control: A $\langle\rangle$ `Bright`. A case in point is that in `L5` we might be constantly brought back to `Off`. For this negative case, we make a "retreat" by assuming that the opponent is not too "hostile". The basic idea is that in order to reach the goal states, we hope that the opponent reacts in favor of us.

The principle of playing games with a cooperative winning strategy is illustrated in Fig. 6. The state space of the timed game is partitioned into three areas: the winning "safe zone", the possibly winning zone, and the losing "no-hope zone". For the weakened test purpose in Section 2.3, it means that if the opponent is willing to cooperate, then we can possibly reach a state, from which the goal location `Bright` is always eventually reachable (the "safe zone").

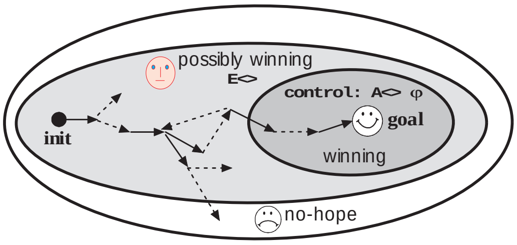


Fig. 6. Playing games with cooperative winning strategies.

Definition 2 (**Cooperative Strategy**). Let $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ be a TIOGA, $(S, s_0, Act_{in}, Act_{out}, \rightarrow)$ be its TIOTS, and $\rightarrow \,=\, \rightarrow_{in} \cup \rightarrow_{out} \cup \rightarrow_d$. A *cooperative strategy* f over $\mathcal{S}$ is defined as a partial function:

$$f: \; S \;\rightarrow\; \{coop\} \times (\rightarrow_{in} \cup \rightarrow_{out} \cup \{\lambda\}) \cup \{winning\} \times (\rightarrow_{in} \cup \{\lambda\}).$$

The projection function f_{stg} indicates which stage ("cooperative" or "winning") f is currently in, and f_{mov} denotes the suggested or desired move of f. For transition $t \in (\rightarrow \setminus \rightarrow_d)$, let $ev(t)$ be the event, and $tgt(t)$ be the target state. In the cooperative stage, if a strategy-desired output occurs as expected, then the opponent is said to be *cooperating*, otherwise the strategy is violated.

Definition 3 (**Supervised Run**). Let $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ be a TIOGA and f a cooperative 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 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:
  - $e \in Act_u$ and $((f_{stg}(last(\sigma)) = winning) \vee ((f_{stg}(last(\sigma)) = coop) \wedge (e = ev(f_{mov}(last(\sigma))))))$,
  - $e \in Act_c$ and $e = ev(f_{mov}(last(\sigma)))$,
  - $e \in \mathbb{R}_{\geq 0}$ and $\forall e' \in [0, e).\exists s'' \in S.((last(\sigma) \xrightarrow{e'} s'') \wedge (f_{mov}(s'') = \lambda))$,
- $\sigma \in \mathsf{SupRuns}(s, f)$ if σ is an infinite run whose finite prefixes are all included in $\mathsf{SupRuns}(s, f)$.

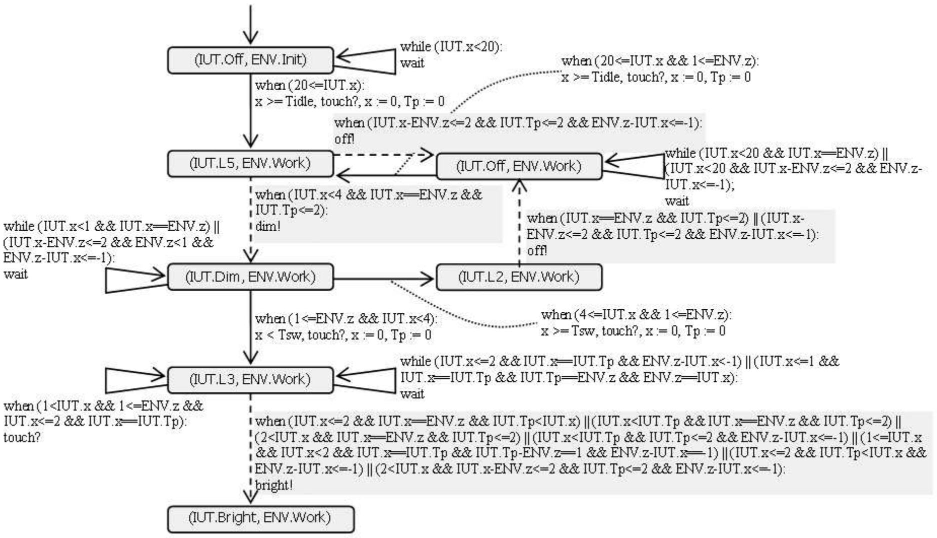
Given a TIOGA $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ and a set of goal states $K \subseteq L \times \mathbb{R}^X$ of its corresponding TIOTS, let $(\mathcal{S}, K)$ be a reachability game. A *maximal run* σ is either an infinite run, or a finite run such that either $last(\sigma) \in K$, or $(last(\sigma) \notin K) \wedge ((last(\sigma) \xrightarrow{\alpha}) \Rightarrow (\alpha = 0))$. A finite or infinite run $\sigma = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots s_n \xrightarrow{\alpha_n} \ldots$ is *winning* if $\exists k \geq 0.(s_k \in K)$. A run σ is *losing* if σ is maximal and $\forall 0 \leq k \leq min\{index(last(\sigma)), \infty\}.(s_k \notin K)$. The set of all maximal runs starting from state s is denoted by $\mathsf{MaxRuns}(s)$, and the set of all winning runs starting from state s is denoted by $\mathsf{WinRuns}(s, \mathcal{S}, K)$.

Definition 4 (**Cooperative Winning Strategy**). Let $\mathcal{S} = (L, l_0, Act, X, E, Inv)$ be a TIOGA, f a cooperative strategy over $\mathcal{S}$, and s 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 *cooperative winning strategy*.

For the TIOGA in Fig. 4 and the weakened test purpose $\mathsf{E}\ \langle\rangle\ \mathsf{control}$: $\mathsf{A}\ \langle\rangle$ `Bright`, UPPAAL-TIGA automatically generates a cooperative winning strategy, as is shown in Fig. 7, where the strategy-desired outputs are in dotted lines.

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



(IUT.Off, ENV.Init)
while (IUT.x<20):
wait
when (20<=IUT.x):
x >= Tidle, touch?, x := 0, Tp := 0
when (20<=IUT.x && 1<=ENV.z):
x >= Tidle, touch?, x := 0, Tp := 0
when (IUT.x-ENV.z<=2 && IUT.Tp<=2 && ENV.z-IUT.x<=-1):
off!
(IUT.L5, ENV.Work)
(IUT.Off, ENV.Work)
while (IUT.x<20 && IUT.x==ENV.z) ||
(IUT.x<20 && IUT.x-ENV.z<=2 && ENV.z-
IUT.x<=-1);
wait
when (IUT.x<4 && IUT.x==ENV.z &&
IUT.Tp<=2):
dim!
while (IUT.x<1 && IUT.x==ENV.z) ||
(IUT.x-ENV.z<=2 && ENV.z<1 &&
ENV.z-IUT.x<=-1):
wait
when (IUT.x==ENV.z && IUT.Tp<=2) || (IUT.x-
ENV.z<=2 && IUT.Tp<=2 && ENV.z-IUT.x<=-1):
off!
(IUT.Dim, ENV.Work)
(IUT.L2, ENV.Work)
when (1<=ENV.z && IUT.x<4):
x < Tsw, touch?, x := 0, Tp := 0
when (4<=IUT.x && 1<=ENV.z):
x >= Tsw, touch?, x := 0, Tp := 0
while (IUT.x<=2 && IUT.x==IUT.Tp && ENV.z-IUT.x<-1) || (IUT.x<=1 &&
IUT.x==IUT.Tp && IUT.Tp==ENV.z && ENV.z==IUT.x):
wait
(IUT.L3, ENV.Work)
when (1<IUT.x && 1<=ENV.z &&
IUT.x<=2 && IUT.x==IUT.Tp):
touch?
when (IUT.x<=2 && IUT.x==ENV.z && IUT.Tp<IUT.x) ||(IUT.x<IUT.Tp && IUT.x==ENV.z && IUT.Tp<=2) ||
(2<IUT.x && IUT.x==ENV.z && IUT.Tp<=2) || (IUT.x<IUT.Tp && IUT.Tp<=2 && ENV.z-IUT.x<=-1) || (1<=IUT.x
&& IUT.x<2 && IUT.x==IUT.Tp && IUT.Tp-ENV.z==1 && ENV.z-IUT.x==1) ||(IUT.x<=2 && IUT.Tp<IUT.x &&
ENV.z-IUT.x<=-1) || (2<IUT.x && IUT.x-ENV.z<=2 && IUT.Tp<=2 && ENV.z-IUT.x<=-1):
bright!
(IUT.Bright, ENV.Work)

Fig. 7. An example cooperative winning strategy.

4 Test Case Generation

A test case for uncontrollable reactive systems should be adaptive rather than sequential preset, thus it should have a tree structure rather than be just a linear I/O sequence. Note that a cooperative winning strategy neither drives the test execution nor issues test verdicts. To have more operational tests, we define a test case as a tree-like TA which permits non-reset clock assignments, and we derive this TA from the TIOGA model $\mathcal{S}$ and the strategy f.

Given a TA location l, we use $outgoing(l)$ to denote the set of output actions originating from l. Given a state s in the TIOTS of a TA, we use $location(s)$ to denote the corresponding location of s in the TA.

Definition 5 (**Test Case**). A *test case* is a TA $\mathcal{T} = (L_t, l_{0t}, Act, X_t, E_t, Inv_t)$ where L_t is a set of locations containing those marked pass, fail and inconc, which are all the terminal nodes, $l_{0t} \in L_t$ is the initial location, $Act = Act_{in} \cup Act_{out}$, X_t is a set of clocks, Inv_t associates to each location its invariant, and E_t is the transition relation such that:

- $\mathcal{T}$ is deterministic,
- $\mathcal{T}$ has bounded behavior, i.e., $\forall \sigma = \sigma_1\sigma_2\sigma_3 \ldots \in \mathsf{Runs}(\mathcal{T}).\exists n > 0.(|\{i | \sigma_i \in Act_{in} \cup Act_{out}\}| < \infty \wedge (\Sigma\{\sigma_i | \sigma_i \in \mathbb{R}_{\geq 0}\}) < n)$,
- $\forall l_t \in (L_t \backslash \{\mathsf{pass}, \mathsf{fail}, \mathsf{inconc}\}).\forall \alpha \in Act_{out}.(\alpha \in outgoing(l_t))$.

The basic idea of test case generation is to keep looking up the generated cooperative winning strategy and the SPEC model to decide when to make what move against the IMP in (forthcoming) test execution, and which decision (pass, fail, inconc, or to continue on by recursively building the test tree) to make upon every possibly observed output from IMP.

Let s be a semantic state of $\mathcal{S}$, s_0 be the initial state. We use $w = width(f_{mov}(s))$ to denote the time width of the observing window of the next move according to the strategy. Let $x \in X_t$ be a unique clock variable for recording the timing constraints in f and for building invariants and transitions in the test case TA. We use another unique clock variable $h \in X_t$ to record the time when a strategy-desired output happens. A location l_t of the test case TA is called a *conditional branching location* if at this location the branching is based on the just-recorded occurrence time h of an output action. Thus l_t is the destination location of the transition of an observed output. Algorithm 4.1 sketches out the main idea of test case derivation. For space reasons we do not elaborate on some tricky parts such as the conditional branching.

A key point of our test generation algorithm is the semantics of forced actions. We adopt the following semantics: if a location invariant of the TIOGA is hit, then we check whether there is some outgoing edge with enabled input action leading to other location, or some self-looping edge with enabled input action and clock resets. If there is no such edge, then we check whether there is some outgoing edge with output action leading to other location, or some self-looping edge with output action and clock resets. If there is also no such edge and the strategy still suggests "delay" when hitting the invariant, then we report fail.

Because the (weakened) test purpose is proved to be satisfiable, the synthesized (cooperative) winning strategy is of finite length and it guides us towards the goal states, Algorithm 4.1 will always terminate. The complexity of this recursive algorithm largely depends on the sizes of the strategy and the Act set.

Fig. 8 gives an example test case TA for the TIOGA in Fig. 4 and the cooperative winning strategy in Fig. 7. There are two inconclusive nodes. The node with invariant $x == 0$ is a conditional branching location.

5 Test Execution

Definition 6 (**Test Execution**). Let $\mathsf{TIOTS}(\mathcal{T}) = (T, t_0, Act_{in}, Act_{out}, \rightarrow_t)$ be the TIOTS of test case $\mathcal{T}$, and assume the IMP may be modeled as another TIOTS $\mathcal{I} = (I, i_0, Act_{in}, Act_{out}, \rightarrow_i)$. The execution of $\mathcal{I}$ with $\mathcal{T}$ is modeled by the synchronous parallel execution $\mathsf{TIOTS}(\mathcal{T})||\mathcal{I}$ which is defined by the rules:

Algorithm 4.1 TestCase($\mathcal{S}$, f)

```
Input: TIOGA specification $\mathcal{S}$, cooperative winning strategy $f$;
Output: a test case TA $\mathcal{T}$;
Initialization: $w := 0; x := 0; h := 0$; add_node($s_0$);
Main: BuildTestCase($s_0$).
Procedure BuildTestCase($s$)  /*$s$: semantic state in $\mathcal{S}$, node in $\mathcal{T}$*/
if $s$ does not correspond to a conditional branching location then
    $w := width(f_{mov}(s))$;
    add invariant "$x \leq w$" for node $s$ in $\mathcal{T}$;
    for each $o \in Act_{out}$ do /* to wake up on every possible output */
        if $o \in outgoing(location(s))$ then
            if the destination state of this transition is a goal state then
                add_edge("$s \xrightarrow{o!}$ pass"); /* to add a node pass and an edge in $\mathcal{T}$ */
            else if $(f_{stg}(s) = coop) \wedge (o \neq ev(f_{mov}(s)))$ then /*not a desired output*/
                add_edge("$s \xrightarrow{o!}$ inconc");
            else /* continue on by recursion on a conditional branching location */
                add_edge("$s \xrightarrow{o!, h:=x, x:=0} s'$"); $s' := (s$ After $h)$ After $o$; BuildTestCase($s'$);
        else
            add_edge("$s \xrightarrow{o!}$ fail");
    end for
    case $f_{mov}(s)$ of /* to delay a period, offer an input, or observe an output */
        "$\lambda$":
            if $((s$ After $w)$ hits invariant of $location(s)) \wedge (f_{mov}(s$ After $w) = \lambda)$ then
                add_edge("$s \xrightarrow{x=w}$ fail"); /* acc. to semantics of forced actions */
            else
                add_edge("$s \xrightarrow{x=w, x:=0} s'$"); $s' := s$ After $w$; BuildTestCase($s'$);
        "to offer input $i$":
            add_edge("$s \xrightarrow{x=w, i?} s'$"); $s' := tgt(f_{mov}(s$ After $w))$; BuildTestCase($s'$);
        "to observe output $o$":
            if $(f_{stg}(s) = coop) \wedge (\{f_{mov}(s)\} \cap \rightarrow_{out} \neq \varnothing) \wedge$ (no output happens) then
                add_edge("$s \xrightarrow{x=w}$ inconc");
    end case
else /* $s$ corresponds to a conditional branching location */
    add invariant "$x = 0$" for $s$; /* an "urgent" location in $\mathcal{T}$ */
    branching according to $f$ and the value of $h$;
    recursive calls of BuildTestCase();
End Procedure
```

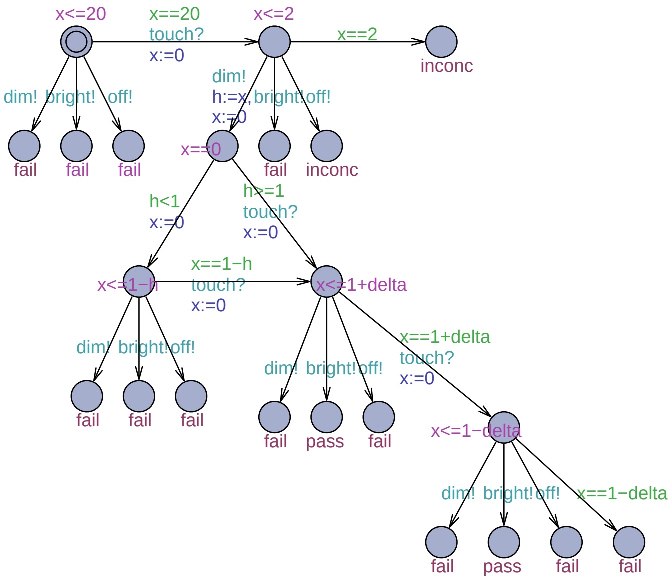


Fig. 8. An example test case TA.

$$\frac{t\xrightarrow{g_1,\alpha}_t t', i\xrightarrow{g_2,\alpha}_i i'}{t||i\xrightarrow{g_1\wedge g_2,\alpha} t'||i'}\ \alpha\in Act_{in},\qquad \frac{t\xrightarrow{\alpha}_t t', i\xrightarrow{\alpha}_i i'}{t||i\xrightarrow{\alpha} t'||i'}\ \alpha\in Act_{out},\qquad \frac{t\xrightarrow{d}_t t', i\xrightarrow{d}_i i'}{t||i\xrightarrow{d} t'||i'}\ d\in\mathbb{R}_{\geq 0}$$

where $t, t'\in T$, $i, i'\in I$, and $g_1, g_2\in\mathcal{C}(X)$.

A test run is a run of the product $\mathsf{TIOTS}(\mathcal{T})||\mathcal{I}$ that leads to a state whose left sub-state corresponds to a terminal node of $\mathcal{T}$. Formally, $\sigma\in\mathsf{Runs}(\mathsf{TIOTS}(\mathcal{T})||\mathcal{I})$ is a *test run* if $\exists i'\in I.\exists t'\in T.((t_0||i_0 \xrightarrow{\sigma} t'||i')\cap(location(t') = \mathsf{pass}\ \vee\ location(t') = \mathsf{fail}\ \vee\ location(t') = \mathsf{inconc}))$. In the pass case we say that $\mathcal{I}$ *passes* test run σ. In the fail case we say that $\mathcal{I}$ *fails* σ. The inconc case indicates neither passing nor failing. It simply means that we do not get cooperation and are thus not assured of reaching the goal states.

Given $\mathcal{I}$ and $\mathcal{T}$, if there is a failing test run of $\mathsf{TIOTS}(\mathcal{T})||\mathcal{I}$, then $\mathcal{I}$ *fails* $\mathcal{T}$. If every test run of $\mathsf{TIOTS}(\mathcal{T})||\mathcal{I}$ is not failing, we say $\mathcal{I}$ *passes* $\mathcal{T}$.

6 Soundness and Completeness

The soundness property of the test method says that if there exists a failing test run, then the system implementation does not conform to the system specification. The partial completeness property of the test method says that if the system implementation does not conform to the system specification with respect to the specified weakened test purpose, then we can always find a failing test run.

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, $\mathcal{I} = (I, i_0, Act_{in}, Act_{out}, \rightarrow_i)$ be a TIOTS implementation, φ be a weakened test purpose such that $\mathcal{S} \models \varphi$, and $\mathcal{S}_f$ be the behavior of $\mathcal{S}$ that are constrained by strategy f, then we have:

Theorem 7 (**Soundness**): $\exists f \in \mathsf{Strategy}(\mathcal{S}, \varphi).(\mathcal{I}\ fails\ \mathsf{TestCase}(\mathcal{S}, f)) \quad \Rightarrow \mathcal{I}\ \cancel{\mathsf{tioco}}\ \mathsf{TIOTS}(\mathcal{S})$.

Proof sketch: Let $\mathcal{T} = \mathsf{TestCase}(\mathcal{S}, f)$ and $\mathsf{TIOTS}(\mathcal{T}) = (T, t_0, Act_{in}, Act_{out}, \rightarrow_t)$. By $(\mathcal{I}\ fails\ \mathcal{T})$ we know that $\exists \sigma \in \mathsf{Runs}(\mathsf{TIOTS}(\mathcal{T})||\mathcal{I}).\exists i' \in I.\exists t' \in T.(t_0||i_0 \xrightarrow{\sigma} t'||i') \cap (location(t') = \mathsf{fail})$. From Algorithm 4.1 we know that there are two cases of finishing with a fail verdict. The first case is that we observe an invalid output w.r.t. $\mathcal{S}$ (lines 12-13 of Alg. 4.1). According to Definition 6 and Definition 1, we conclude that $\mathcal{I}\ \cancel{\mathsf{tioco}}\ \mathsf{TIOTS}(\mathcal{S})$. The second case is when we are hitting the location invariant (lines 17-18). According to the forced semantics of controllable and uncontrollable actions in this circumstance, there should be a forced output. But unfortunately we have not observed it. Thus the conformance relation has been violated. Therefore comes $\mathcal{I}\ \cancel{\mathsf{tioco}}\ \mathsf{TIOTS}(\mathcal{S})$. $\square$

Theorem 8 (**Partial Completeness**): $\exists f_1 \in \mathsf{Strategy}(\mathcal{S}, \varphi).(\mathcal{I}\ \cancel{\mathsf{tioco}}\ \mathcal{S}_{f_1}) \quad \Rightarrow \exists f_2 \in \mathsf{Strategy}(\mathcal{S}, \varphi).(\mathcal{I}\ fails\ \mathsf{TestCase}(\mathcal{S}, f_2))$.

Proof sketch: By $(\mathcal{I}\ \cancel{\mathsf{tioco}}\ \mathcal{S}_{f_1})$ we know that there exists a timed trace σ such that $\sigma \in \mathsf{TTr}(\mathcal{I})$ and $\sigma \notin \mathsf{TTr}(\mathcal{S}_{f_1})$ according to Definition 1. For simplicity, we suppose σ ends with the first violation w.r.t. $\mathcal{S}_{f_1}$. According to Algorithm 4.1 we know that this has two possible consequences. The first case is that σ has an output action which is disallowed in $\mathsf{TIOTS}(\mathcal{S})$. The second case is that σ has an observed quiescence when hitting a location invariant, but it is disallowed in $\mathsf{TIOTS}(\mathcal{S})$. Therefore we can build another timed trace σ' such that σ' has exactly the same prefix as σ, but σ' ends without a violation w.r.t. $\mathsf{TIOTS}(\mathcal{S})$. Therefore, we can generate some cooperative winning strategy f_2 from $\mathcal{S}$ and φ, and build a test case TIOTS from $\mathcal{S}$ and f_2 such that σ' is not a failing run but σ is a failing run. According to Definition 6, we can conclude that $\exists f_2 \in \mathsf{Strategy}(\mathcal{S}, \varphi).(\mathcal{I}\ fails\ \mathsf{TestCase}(\mathcal{S}, f_2))$. $\square$

7 Case Study

We consider a simple Leader Election Protocol (LEP) problem by Leslie Lamport, where we have one TIOGA for an arbitrary node (the “plant”), and two TA for simulating all other nodes and a buffer with certain capacity (the “controller”). The TIOGA has uncontrollable actions in the sense that in the plant node a *timeout*! event might occur after waiting for a certain period of time without receiving “useful” messages, and an *ignore*! event might occur due to loss of messages. More details can be found in a forthcoming technical report on the authors’ homepages. We defined the following test purposes:

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

All these test purposes are checked to be false using UPPAAL-TIGA. However, all the weakened test purposes (prefixed with "`E ⟨⟩`") are checked to be true. 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 of CPU time overheads and the memory consumptions, where / means "out of memory". Each sub-column 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 $(n-1)$. The table indicates that for some test purposes, cooperative winning strategy generation for the LEP protocol with up to 7 nodes takes less than 10 minutes and the memory consumption is not well beyond our expectation considering the complexity of the problem. Since strategy generation is the most computation intensive step in our test framework, our testing method seems not to be only of theoretical value.

Table 1. Cooperative winning strategy generation for LEP with lossy channels.

	Time (s)						**Memory** (MB)					
	n=3	4	5	6	7	8	n=3	4	5	6	7	8
TP1	0.04	0.17	0.81	3.21	10.57	30.65	0.1	4.2	7.9	18.9	48.6	119.5
TP2	0.11	1.32	11.74	85.14	558.67	/	4.3	13.0	80.3	517.0	2958.9	/
TP3	3.22	75.56	/	/	/	/	24.3	493.5	/	/	/	/

8 Conclusions

We examine black-box conformance testing of uncontrollable timed systems using a cooperative game-based approach. We model the systems with Timed I/O Game Automata and specify the test purposes with TCTL formulas. We generate cooperative winning strategies, derive test cases, and execute them on the implementation. The test method is proved to be sound and complete w.r.t. the test purpose. Preliminary experimental results indicate that it is a viable approach. This opens up the possibility of testing a broader type of properties on uncontrollable TA models that are previously thought of as not testable.

Future work include: 1) more case studies for performance evaluation, test effectiveness analysis and method scalability improvement; 2) generalizing state-based strategy to history-based strategy; 3) implementing the test case generation and execution algorithm to build a fully automated strategy-based testing environment; 4) strategy-based testing with partial observability.

References

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

2. Behrmann, G., Cougnard, A., David A., Fleury E., Larsen, K.G., Lime, D.: UPPAAL-TIGA: Time for Playing Games! In: CAV'07, Springer (2007) 121–125
3. Brandán Briones, L., Brinksma, E.: A Test Generation Framework for quiescent Real-Time Systems. In: FATES'04, Springer (2004) 64–78
4. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In: CONCUR'05, Springer (2005) 66–80
5. Clarke, D., Lee, I.: Testing real-time constraints in a process algebraic setting. In: ICSE'95, (1995) 51–60
6. David, A., Larsen, K.G., Li, S., Nielsen, B.: A Game-Theoretic Approach to Real-Time System Testing. In: DATE'08, to appear
7. En-Nouaary, A., Dssouli, R., Khendek, F., Elqortobi, A.: Timed Test Cases Generation Based on State Characterization Technique. In: RTSS'98, (1998) 220–229
8. Hessel, A., Larsen, K.G., Nielsen, B., Pettersson, P., Skou, A.: Time-Optimal Real-Time Test Case Generation Using UPPAAL. In: FATES'03, Springer (2003) 114–130
9. Higashino, T., Nakata, A., Taniguchi, K., Cavalli, A.R.: Generating Test Cases for a Timed I/O Automaton Model. In: IWTCS'99, Kluwer (1999) 197–214
10. Jard, C., Jéron, T.: TGV: theory, principles and algorithms. Software Tools for Technology Transfer (STTT), **7**(4) 297–315
11. Khoumsi, A., Jéron, T., Marchand, H.: Test Cases Generation for Nondeterministic Real-Time Systems. In: FATES'03, Springer (2003) 131–146
12. Krichen, M., Tripakis, S.: Black-box conformance testing for real-time systems. In: SPIN'04, Springer (2004) 109–126
13. Larsen, K.G., Mikucionis, M., Nielsen, B.: Online Testing of Real-time Systems Using UPPAAL. In: FATES'04, Spinger (2004) 79–94
14. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: STACS'95, Spinger (1995) 229–242
15. Nunez, M., Rodriguez, I.: Conformance Testing Relations for Timed Systems. In: FATES'05, Spinger (2005) 103–117
16. Springintveld, J., Vaandrager, F.W., D'Argenio, P.R.: Testing timed automata. Theoretical Computer Science, **254**(1-2) (2001) 225–257
17. Tretmans, J.: Testing concurrent systems: a formal approach. In: CONCUR'99, Spinger (1999) 46–65
18. Yannakakis, M.: Testing, optimization, and games. In: ICALP'04, (2004) 28–45