Testing Timed Systems Using Determinization Techniques for One-Clock Timed Automata

Moez Krichen[1,2]

[1]Faculty of Computer Science and Information Technology, Al-Baha University, KSA
[2]ReDCAD Research Laboratory, University of Sfax, Tunisia

Abstract

In this work, we are interested in formal Model-Based Testing for Real-Time Systems. The proposed approach is based on the use of the model of Timed Automata with continuous clocks for which we adopt the reset-point semantics. We remind the definition of timed conformance relation tioco. We extend the notion of soundness and completeness of test suites. We also consider specifications in form of one-clock input-complete timed automata. Moreover, we provide interesting decidability results for the considered classes of specifications. More specifically, we consider the case when some parameters of the timed-automaton tester are fixed in advance, namely the number of clocks of the timed-automaton and its maximal time-constraint constants. Finally, several possible extensions of the present work in different directions are proposed.

Keywords

Model-Based Testing (MBT) | Formal Methods (FM) | Real-Time Testing (RTT) | Determinization Techniques (DT) | One-Clock Timed Automata (OC-TA)

1. Introduction

In this work we are intersted in Model-Based Testing (MBT) for Real-Time Systems [1, 2]. This technique consists in describing the behavior of the System Under Test (SUT) using a specific adequate formalism and then producing automatically test scenarios from the available descriptions with respect to some selection criteria adopting some coverage methods. The next phase consists in running the obtained tests suites on the SUT and calculating the corresponding verdicts in order to check whether the implementation conforms to its model or not. This paper extends some of our previous contributions [3, 4, 5] about MBT for real-time systems. These works were mainly built on the classical timed automaton model [6].

Timed Automata (TA) [7] model is one of the most well-known mathematical formalism for designing real-time systems. This model can be seen as an extension of finite automata with continuous clocks which may be used to guarantee the correctness of some timed-constraints. Many tools based on this model were developped during the last few years, namely: UPPAAL [8], PRISM [9], UPPAAL Tiga [10], etc.

Tunisian Algerian Conference on Applied Computing (TACC 2021), December 18–20, 2021, Tabarka, Tunisia
moez.krichen@redcad.org (M. Krichen)
https://www.redcad.org/members/mkrichen/ (M. Krichen)
0000-0001-8873-9755 (M. Krichen)

Deterministic Timed Automata represent a specific type of timed automata which has stronger properties. This class of timed automata may be used in different fields like learning [11, 12], fault diagnosis [13], test generation [14], etc. In general, it is not possible to convert a given non-deterministic timed automaton to an equivalent deterministic timed automata. However, there are some specific classes of timed automata which are *determinisable* such as: timed automata with integer-resets [15], event-clock timed automata [16], strongly non-zeno timed automata [17], etc. For the case where determinization [18] is not possible, it may be possible to use some approximation techniques like the ones proposed in [19, 20, 21] in the context of model-based testing of real-time systems.

Our new proposed approach is mainly inspired by [22]. We adopt the *reset-point semantics* for timed-automata for model-based testing purposes. We adapt the definition of our timed input-output conformance relation tioco with respect to the new considered semantics. We extend the notion of soundness and completeness of test suites correspondingly. We also consider the case of specifications given as one-clock input-complete timed automata for which interesting decidability properties exist. More precisely, we consider the case when some parameters of the timed-automaton tester are fixed in advance (e.g., number of clocks and maximal constants).

Next in Section 2 we recall some fundamentals about timed languages and clock-constraints. In Section 3 we give details about the timed automaton model and the underlying semantics. Section 4 introduces the adopted testing framework. Section 5 summarizes the most important results. Finally, Section 6 concludes the article and proposes some directions for future work.

2. Fundamentals

We adopt almost the same definitions and notations as in [22].

2.1. Timed Languages

Consider:

- $\mathtt{ACT}$: a nonempty finite set of discrete actions;
- $\mathbb{R}eals$: the set of reals;
- $\mathbb{R}eals_{\geq 0}$: the set of nonnegative reals.

A *timed word* tw over the set of actions $\mathtt{ACT}$ is of the form:
$tw = (act_1, time_1) \ldots (act_n, time_n) \in (\mathtt{ACT} \times \mathbb{R}eals)^*$ such that the sequence of instants $time_i$ satisfy the following:

$$0 \leq time_1 \leq time_2 \leq \cdots \leq time_n.$$

Given two timed words tw_1 and tw_2 such that:

- $tw_1 = (act_1, time_1) \ldots (act_n, time_n)$;
- $tw_2 = (act_{n+1}, time_{n+1}) \ldots (act_{n+m}, time_{n+m})$;

- $time_{n+1} \geq time_n$.

The concatenation of tw_1 and tw_2 denoted $tw_1 \cdot tw_2$ is defined as:

$$tw_1 \cdot tw_2 = (act_1, time_1) \ldots (act_n, time_n)(act_{n+1}, time_{n+1}) \ldots (act_{n+m}, time_{n+m}).$$

We also define the following entities:

- $\mathbb{TW}(\texttt{ACT})$: the set of timed words over Act;
- For $time \in \mathsf{R}$, $\mathbb{TW}_{\geq time}(Act)$: the set of timed words such that $time_1 \geq time$;
- A *timed language* TL over the set of actions Act is a subset of $\mathbb{TW}(Act)$, i.e.: $TL \subseteq \mathbb{TW}(Act)$;
- For $tw = (act_1, time_1) \ldots (act_n, time_n) \in \mathbb{TW}(\texttt{ACT})$ and $TL \subseteq \mathbb{TW}(\texttt{ACT})$:

$$tw^{-1}TL := \left\{ tw' \in \mathbb{TW}(Act) \;\middle|\; tw \cdot tw' \in TL \right\};$$

  In this case, we clearly have: $tw^{-1}TL \subseteq \mathbb{TW}_{\geq time_n}(\texttt{ACT})$.

2.2. Clock constraints

Consider a finite set of clocks:

$$\texttt{CLOCKS} = \{\mathsf{c}clock_1, \ldots, \mathsf{c}clock_k\}.$$

A *clock valuation* is a function:

$$val \in \mathbb{R}eals_{\geq 0}^{\texttt{CLOCKS}}$$

assigning a non-negative real number $val(clock)$ to every clock $\mathsf{c}clock \in \texttt{CLOCKS}$.

A *clock constraint* is a formula of the form:

$$\varphi = \mathsf{true} \mid \mathbf{false} \mid \mathsf{c}clock_i - \mathsf{c}clock_j \;\texttt{cmp}\; const \mid \mathsf{c}clock_i \;\texttt{cmp}\; const \mid \varphi_1 \wedge \varphi_2$$

such that $\texttt{cmp} \in \{<, \leq, =, >, \geq\}$ is a comparison operator and $const \in \mathsf{N}$.[1]

We say that the valuation val satisfies the constraint φ if interpreting every clock $\mathsf{c}clock_i$ by $val(\mathsf{c}clock_i)$ makes the clock constraint φ a tautology. In this case, we will use the following notation:

$$val \models \varphi.$$

We also define the set of valuations satisfying φ as follows:

$$[\![\varphi]\!] = \{val \in \mathbb{R}eals_{\geq 0}^{\texttt{CLOCKS}} \mid val \models \varphi\}.$$

[1] N being the set of non-negative integers.

3. Timed automata

A *timed automaton* (TA) is a tuple

$$\texttt{TA} = (Act, \texttt{LOC}, \texttt{CLOCKS}, \texttt{INI}, \texttt{FIN}, \texttt{EDGES})$$

such that:

- ACT: finite set of discrete actions;
- LOC: finite set of locations;
- CLOCKS: finite set of continuous clocks;
- $\texttt{INI} \subseteq \texttt{LOC}$: set of initial locations;
- $\texttt{FIN} \subseteq \texttt{LOC}$: set of final locations;
- EDGES: finite set of edges.

The edges in EDGES are of the form:

$$edg = (loc_i, act, \varphi, \texttt{rst}, loc_j)$$

such that:

- $loc_i \in \texttt{LOC}$: source location of the transition;
- $loc_j \in \texttt{LOC}$: destination location of the transition;
- $act \in \texttt{ACT}$: discrete action labeling the transition;
- φ: clock constraint that must be true for allowing the transition to be executed;
- $\texttt{rst} \subseteq \texttt{CLOCKS}$: the set of clocks to be reset after the execution of the transition.

Next, we introduce the so-called *reset-point semantics* [23, 24] for timed automata. A *configuration* of a timed automaton $\texttt{TA} = (Act, \texttt{LOC}, \texttt{CLOCKS}, \texttt{INI}, \texttt{FIN}, \texttt{EDGES})$ is a tuple:

$$(loc, val, current\_time)$$

such that:

- $loc \in \texttt{LOC}$;
- $val \in \mathbb{R}eals^{\texttt{CLOCKS}}$;
- $current\_time \in \mathbb{R}eals$;
- $\forall\ clock \in \texttt{CLOCKS} : val(clock) \leq current\_time$.

A configuration is said to be *initial* if and only if:

- $loc \in \texttt{INI}$;
- $current\_time = 0$;
- $\forall\ \texttt{c}clock \in \texttt{CLOCKS} : val(\texttt{c}clock) = 0$.

A configuration is said to be *final* if and only if:

- $loc \in \texttt{FIN}$.

The set of all possible configurations of TA is denoted:

$$\texttt{CONFIG(TA)}.$$

For $val \in \mathbb{R}eals^{\texttt{CLOCKS}}$, $\texttt{rst} \subseteq \texttt{CLOCKS}$ and $time \in \mathbb{R}eals$, we define the valuation

$$val^{[\texttt{rst} \mapsto time]} \in \mathbb{R}eals^{\texttt{CLOCKS}}$$

as follows:

- $\forall\ \texttt{c}clock \in \texttt{rst}$: $val^{[\texttt{rst} \mapsto time]}(\texttt{c}clock) = time$;
- $\forall\ \texttt{c}clock \in \texttt{CLOCKS} \setminus \texttt{rst}$: $val^{[\texttt{rst} \mapsto time]}(clock) = val(\texttt{c}clock)$.

Similarly for $val \in \mathbb{R}eals^{\texttt{CLOCKS}}$ and $time \in \mathbb{R}eals$, we define the valuation

$$val^{time} \in \mathbb{R}eals^{\texttt{CLOCKS}}$$

as follows:

- $\forall\ \texttt{c}clock \in \texttt{CLOCKS}$: $val^{time}(\texttt{c}clock) = time - val(\texttt{c}clock)$.

For the edge $edg = (loc_i, act, \varphi, \texttt{rst}, loc_j) \in \texttt{EDGES}$, $val_i \in \mathbb{R}eals^{\texttt{CLOCKS}}$ and $time \in \mathbb{R}eals$, we define the *transition*:

$$(loc_i, val_i, current\_time) \xrightarrow{act,time} (loc_j, val_j, time)$$

such that:

- $val_j \in \mathbb{R}eals^{\texttt{CLOCKS}}$;
- $time \geq current\_time$;
- $val_i^{time} \models \varphi$;
- $val_j = val_i{}^{[\texttt{rst} \mapsto time]}$.

The set of all possible transitions of TA is denoted:

$$\texttt{TRANS(TA)}.$$

The timed automaton TA induces a *timed labeled transition system* $\texttt{TLTS(TA)}$ defined as follows:

$$\texttt{TLTS(TA)} = (\texttt{ICONF(TA)}; \texttt{CONFIG(TA)}, \texttt{TRANS(TA)}, \texttt{FCONF(TA)})$$

such that:

- $\texttt{ICONF}(\texttt{TA})$ is the set of all possible initial configurations;
- $\texttt{FCONF}(\texttt{TA})$ is the set of all possible final configurations.

Given the timed word:

$$tw = (act_1, time_1) \ \ldots \ (act_n, time_n) \in \mathbb{TW}(\texttt{ACT})$$

and the set of configurations:

$$(conf_i = (loc_i, val_i, time_i))_{0 \leq i \leq n} \subseteq \texttt{CONFIG}(\texttt{TA})$$

we say that the sequence:

$$tp = conf_0 \xrightarrow{act_1, time_1} conf_1 \ldots conf_{n-1} \xrightarrow{act_n, time_n} conf_n$$

is a *timed path* of the timed automaton $\texttt{TA}$ if for each $0 \leq i \leq n-1$:

$$conf_i \xrightarrow{act_1, time_1} conf_{i+1} \in \texttt{TRANS}(\texttt{TA}).$$

The timed path tp is said to be *accepted* by the timed automaton $\texttt{TA}$ if:

$$conf_n \in \texttt{FCONF}(\texttt{TA}).$$

In this case, we will use the following notation:

$$conf_0 \xRightarrow{tw} conf_n.$$

In case $conf_0 \in \texttt{ICONF}(\texttt{TA})$, we may also write:

$$\texttt{TA} \xRightarrow{tw} conf_n.$$

Consider the configuration $conf = (loc, val, time) \in \texttt{CONFIG}(\texttt{TA})$ and the timed word: $tw \in \mathbb{TW}(\texttt{ACT})$. We will use the notation: $conf \xRightarrow{tw}$ in case there exists a configuration $conf' \in \texttt{FCONF}(\texttt{TA})$ such that: $conf \xRightarrow{tw} conf'$. The timed language *recognised* by the configuration $conf \in \texttt{CONFIG}(\texttt{TA})$ with respect to the timed automaton $\texttt{TA}$ is defined as:

$$\texttt{TLang}_{\texttt{TA}}(conf) = \left\{ tw \in \mathbb{TW}(\texttt{ACT}) \;\middle|\; conf \xRightarrow{tw} \right\}.$$

Similarly, the timed language recognised by the timed automaton $\texttt{TA}$ is defined as:

$$\texttt{TLang}(\texttt{TA}) = \bigcup_{conf \in \texttt{ICONF}(\texttt{TA})} \texttt{TLang}_{\texttt{TA}}(conf).$$

The timed automaton $\texttt{TA}$ is called *empty timed automaton* when it recognizes the empty language. That is:

$$\texttt{TLang}(\texttt{TA}) = \emptyset.$$

Similarly, it is called *full timed automaton* if it accepts all possible timed words. That is:

$$\texttt{TLang}(\texttt{TA}) = \mathbb{TW}(\texttt{ACT}).$$

A timed automaton TA is said to be *deterministic* if it has only one initial location and, for every two edges $(loc_1, act, \varphi, \texttt{rst}, loc_2)$, $(loc'_1, act', \varphi', \texttt{rst}', loc'_2) \in \texttt{EDGES}$, if $loc_1 = loc'_1$, $act = act'$ and $[\![\varphi \wedge \varphi']\!] \neq \emptyset$ then $\texttt{rst} = \texttt{rst}'$ and $loc_2 = loc'_2$. A *One-Clock Timed Automaton* is a timed automaton which has only one clock.

4. Testing Framework

Starting from this section, we will assume that the set of actions ACT is equal to the union of two disjoint sets $\texttt{ACT}_\texttt{I}$ and $\texttt{ACT}_\texttt{O}$ which are respectively the set of input-actions and output-actions. That is:

$$\texttt{ACT} = \texttt{ACT}_\texttt{I} \cup \texttt{ACT}_\texttt{O} \text{ and } \texttt{ACT}_\texttt{I} \cap \texttt{ACT}_\texttt{O} = \emptyset.$$

A given timed automaton TA is said to be *input-complete* if for every configuration $conf = (loc, val, time) \in \texttt{CONFIG}(\texttt{TA})$ and each pair $(act, time') \in \texttt{ACT}_\texttt{I} \times \mathbb{R}eals_{\geq 0}$ we have:

$$conf \xRightarrow{(act, time+time')} .$$

4.1. Conformance Relation

Consider a timed automaton TA and a timed word $tw \in \mathbb{TW}(\texttt{ACT})$, TA after tw is the set of configurations of $\mathcal{A}$ which may be reached after the execution of tw. Mathematically:

$$\texttt{TA after } tw = \{conf \in \texttt{CONFIG}(\texttt{TA}) \mid \texttt{TA} \xRightarrow{tw} conf\}.$$

Given the configuration $conf = (loc, val, time) \in \texttt{CONFIG}(\texttt{TA})$, $\mathsf{outputs}(conf)$ is the set of all outputs that may be produced when the system is occupying configuration $conf$. Mathematically:

$$\mathsf{outputs}(conf) = \{(act, time') \in \texttt{ACT}_\texttt{O} \times \mathbb{R}eals_{\geq 0} \mid conf \xRightarrow{(act, time+time')}\}$$

The definition is extended naturally to a set of configurations $Conf$.

$$\mathsf{outputs}(Conf) = \bigcup_{conf \in Conf} \mathsf{outputs}(conf).$$

Given two timed automata $\mathcal{S}pec$ (specification) and $\mathcal{I}mp$ (implementation) defined with respect to the same sets of inputs and outputs The relation tioco [19, 25] is defined as follows:

$\mathcal{I}mp$ tioco $\mathcal{S}pec$ *iff* $\forall tw \in \texttt{TLang}(\mathcal{S}pec)$: $\mathsf{outputs}(\mathcal{I}mp \text{ after } tw) \subseteq \mathsf{outputs}(\mathcal{S}pec \text{ after } tw)$.

The relation means that the implementation $\mathcal{I}mp$ conforms to the specification $\mathcal{S}pec$ if and only if for every timed word tw of $\mathcal{S}pec$, the set of outputs of $\mathcal{I}mp$ after the execution of tw is a subset of the set of outputs that can be generated by $\mathcal{S}pec$.

4.2. Timed Test Cases

A timed test scenario for the specification $\mathcal{S}pec$ over ACT is a total function

$$TTS : (\mathbb{R}_{\geq 0} \cup \mathtt{ACT})^* \rightarrow \mathtt{ACT_I} \cup \{\mathsf{WAITING}, \mathsf{SUCCESS}, \mathsf{REJECT}\}.$$

$TTS(tw)$ indicates the action that must be executed by the tester once it observes tw. If $TTS(tw) = inp \in \mathtt{ACT_I}$ then the tester produces input inp. If $TTS(tw) = \mathsf{WAITING}$ then the tester lets time elapse (waits). If $TTS(tw) \in \{\mathsf{SUCCESS}, \mathsf{REJECT}\}$ then the tester produces a verdict and stops.

The execution of TTS on $\mathcal{I}mp$ may be seen as the *parallel composition* of the TLTS defined by TTS and $\mathcal{I}mp$. This composition is denoted by $\mathcal{I}mp \| TTS$. Formally, we will announce that the implementation $\mathcal{I}mp$ *passes* TTS, denoted $\mathcal{I}mp$ pass TTS, if state REJECT may not be reached in $\mathcal{I}mp \| TTS$. We conclude that the implementation passes (respectively fails) the test suite $\mathcal{TST}$ if it passes all tests (respectively fails at least one test) in $\mathcal{TST}$. $\mathcal{TST}$ is said to be *sound with respect to* $\mathcal{S}pec$ if

$$\forall \mathcal{I}mp \ : \ \mathcal{I}mp \ \mathsf{tioco} \ \mathcal{S}pec \Rightarrow \mathcal{I}mp \ \mathsf{pass} \ \mathcal{TST}.$$

Similalry $\mathcal{TST}$ is said to be *complete with respect to* $\mathcal{S}pec$ if

$$\forall \mathcal{I}mp \ : \ \mathcal{I}mp \ \mathsf{pass} \ \mathcal{TST} \Rightarrow \mathcal{I}m \ \mathsf{tioco} \ \mathcal{S}pec.$$

The timed test suite $\mathcal{TST}$ is said to be *exact* with respect to $\mathcal{S}pec$ if it is both sound and complete with respect to $\mathcal{S}pec$.

Our goal is then to generate test suites which are both sound and complete. More specifically, our goal is to produce timed test scenarios which are finitely representable in the form of deterministic timed automata.

5. Main Results

In this section, we assume that the specification we have in hands is given as a non-deterministic timed automaton $\mathcal{S}pec$ which is input-complete and we aim to generate a timed tester corresponding to this specification and which is represented using a deterministic timed automaton TTS which is input-complete and which has one or more clocks. Next, we list some interesting results about this timed automaton tester.

We first start with two intuitive rules related to the cases when the timed automaton tester is empty and full respectively.

Lemma 1. *If TTS is empty then it is complete with respect to the specification $\mathcal{S}pec$.*

Lemma 2. *If TTS is full then it is sound with respect to the specification $\mathcal{S}pec$.*

Now, we consider the situation when the timed automaton tester TTS is, respectively, an under-approximation and an over-approximation of the specification $\mathcal{S}pec$.

Lemma 3. *If $\mathtt{TLang}(TTS) \subseteq \mathtt{TLang}(\mathcal{S}pec)$ (i.e., TTS under-approximation of $\mathcal{S}pec$) then TTS is complete with respect to the specification $\mathcal{S}pec$.*

Lemma 4. *If* $\texttt{TLang}(\mathcal{S}pec) \subseteq \texttt{TLang}(TTS)$ *(i.e.,* TTS *over-approximation of* $\mathcal{S}pec$*) then* TTS *is sound with respect to the specification* $\mathcal{S}pec$*.*

Consequently, we may deduce the following result.

Lemma 5. *If* $\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS)$ *then* TTS *is exact with respect to the specification* $\mathcal{S}pec$*.*

Clearly, Lemma 1 (respectively, Lemma 2) can be seen as a particular case of Lemma 3 (respectively, Lemma4).

Next we consider the following list of problems and we check their decidability.

(P1) Given a specification presented as a non-deterministic timed automaton $\mathcal{S}pec$ which has **two or more clocks**, does it exist a deterministic timed automaton tester TTS such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

(P2) Given a specification presented as a non-deterministic timed automaton $\mathcal{S}pec$ which has **two or more clocks** and given a non-negative integer max_clock, does it exist a deterministic timed automaton tester TTS which has less than max_clock clocks and such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

(P3) Given a specification presented as a non-deterministic timed automaton $\mathcal{S}pec$ which has **two or more clocks** and given two non-negative integers max_clock and $max\_constant$ does it exist a deterministic timed automaton tester TTS which has less than max_clock clocks, the numerical constants of which are equal or smaller than $max\_constant$ and such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

Lemma 6. *The three above problems (P1), (P2) and (P3) are undecidable.*

Next, we consider three similar problems for the case of one-clock non-deterministic timed automata with epsilon transitions.

(P4) Given a specification presented as a **one-clock** non-deterministic timed automaton **with epsilon transitions** $\mathcal{S}pec$ which has **two or more clocks**, does it exist a deterministic timed automaton tester TTS such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

(P5) Given a specification presented as a **one-clock** non-deterministic timed automaton **with epsilon transitions** $\mathcal{S}pec$ and a non-negative integer max_clock, does it exist a deterministic timed automaton tester TTS which has less than max_clock clocks and such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

(P6) Given a specification presented as a **one-clock** non-deterministic timed automaton **with epsilon transitions** $\mathcal{S}pec$ and two non-negative integers max_clock and $max\_constant$, does it exist a deterministic timed automaton tester TTS which has less than max_clock clocks, the numerical constants of which are equal or smaller than $max\_constant$ and such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

Lemma 7. *The three problems (P4), (P5) and (P6) are undecidable too.*

Next, we consider the situation where the specification $\mathcal{S}pec$ is a one-clock timed automaton without epsilon transitions.

(P7) Given a specification presented as a **one-clock** non-deterministic timed automaton **without epsilon transitions** $\mathcal{S}pec$ which has **two or more clocks**, does it exist a deterministic timed automaton tester TTS such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

(P10) Given a specification presented as a **one-clock** non-deterministic timed automaton **without epsilon transitions** $\mathcal{S}pec$ and a positive integer $max\_constant$, does it exist a deterministic timed automaton tester TTS the numerical constants of which are equal or smaller than $max\_constant$ and such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

(P9) Given a specification presented as a **one-clock** non-deterministic timed automaton **without epsilon transitions** $\mathcal{S}pec$ and a non-negative integer max_clock, does it exist a deterministic timed automaton tester TTS which has less than max_clock clocks and such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

(P10) Given a specification presented as a **one-clock** non-deterministic timed automaton **without epsilon transitions** $\mathcal{S}pec$ and two non-negative integers max_clock and $max\_constant$, does it exist a deterministic timed automaton tester TTS which has less than max_clock clocks, the numerical constants of which are equal or smaller than $max\_constant$ and such that:

$$\texttt{TLang}(\mathcal{S}pec) = \texttt{TLang}(TTS).$$

Lemma 8. *The problems (P7) and (P8) are undecidable while the problems (P9) and (P10) are decidable.*

The positive result mentioned in the second part of Lemma 8 may be useful for building exact timed automata testers for the considered class of specifications (one-clock timed automata without epsilon transitions). For the other situations where the considered problems are undecidable, we may use approximation techniques to construct either sound or complete timed automata testers which are as precise as possible.

6. Conclusion and Future Work

In this work, we presented a formal testing framework for real-time systems based on the model of timed automata and the use of the reset-point semantics. Some interesting results were identified. These results may represent a starting point for many future extensions:

- First, making some experimental work for developing timed testers for the case of specifications presented as one-clock timed automata without epsilon transitions.
- Second, identifying some optimal approximation techniques for generating timed testers which are either sound or complete.
- Third, considering the case where the specification of the system under test is given as a product of a set of timed automata.
- Fourth, considering other types of restrictions on the structure and the size of the timed testers we aim to produce such as the number of locations, the number of edges, etc.
- Fifth, considering some adequate selection criteria for generating timed testers with reasonable size and which guarantee optimal coverage of the considered specification.

References

[1] J. Tretmans, Testing concurrent systems: A formal approach, in: Proceedings of the 10th International Conference on Concurrency Theory, CONCUR '99, Springer-Verlag, London, UK, UK, 1999, pp. 46–65. URL: http://dl.acm.org/citation.cfm?id=646734.701460.

[2] M. Krichen, Contributions to model-based testing of dynamic and distributed real-time systems, Ph.D. thesis, École Nationale d'Ingénieurs de Sfax (Tunisie), 2018.

[3] M. Krichen, Testing real-time systems using determinization techniques for automata over timed domains, in: International Colloquium on Theoretical Aspects of Computing, Springer, Cham, 2019, pp. 124–133.

[4] M. Krichen, A formal framework for black-box conformance testing of distributed real-time systems, International Journal Critical Computer Based Systems 3 (2012) 26–43. URL: http://dx.doi.org/10.1504/IJCCBS.2012.045075. doi:10.1504/IJCCBS.2012.045075.

[5] M. Krichen, A formal framework for conformance testing of distributed real-time systems, in: OPODIS, volume 6490 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 139–142.

[6] R. Alur, D. Dill, The theory of timed automata, in: J. W. de Bakker, C. Huizing, W. P. de Roever, G. Rozenberg (Eds.), Real-Time: Theory in Practice, Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, pp. 45–73.

[7] R. Alur, D. L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (1994) 183–235.

[8] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi, M. Hendriks, Uppaal 4.0, in: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems, QEST '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 125–126. URL: https://doi.org/10.1109/QEST.2006.59. doi:10.1109/QEST.2006.59.

[9] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: G. Gopalakrishnan, S. Qadeer (Eds.), Proc. of CAV'11, volume 6806 of *LNCS*, Springer, 2011, pp. 585–591.

[10] F. Cassez, A. David, E. Fleury, K. G. Larsen, D. Lime, Efficient on-the-fly algorithms for the analysis of timed games, in: M. Abadi, L. de Alfaro (Eds.), Proc. of CONCUR'05, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 66–80.

[11] S. Verwer, M. de Weerdt, C. Witteveen, An algorithm for learning real-time automata, in: Proc of. the Annual Belgian-Dutch Machine Learning Conference (Benelearn'078), 2007.

[12] M. Tappler, B. K. Aichernig, K. G. Larsen, F. Lorber, Time to learn - learning timed automata from tests, in: É. André, M. Stoelinga (Eds.), Proc. of FORMATS'19, Springer International Publishing, Cham, 2019, pp. 216–235.

[13] P. Bouyer, F. Chevalier, D. D'Souza, Fault diagnosis using timed automata, in: Proc. of FOSSACS'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 219–233. doi:10.1007/978-3-540-31982-5_14.

[14] B. Nielsen, A. Skou, Automated test generation from timed automata, International Journal on Software Tools for Technology Transfer 5 (2003) 59–77. URL: https://doi.org/10.1007/s10009-002-0094-1. doi:10.1007/s10009-002-0094-1.

[15] P. V. Suman, P. K. Pandya, S. N. Krishna, L. Manasa, Timed automata with integer resets: Language inclusion and expressiveness, in: Proc. of FORMATS'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 78—92. URL: https://doi.org/10.1007/978-3-540-85778-5_7. doi:10.1007/978-3-540-85778-5_7.

[16] R. Alur, L. Fix, T. A. Henzinger, Event-clock automata: a determinizable class of timed automata, Theor. Comput. Sci. 211 (1999) 253–273.

[17] E. Asarin, O. Maler, A. Pnueli, J. Sifakis, Controller synthesis for timed automata, in: Proc. of the 5th IFAC Conference on System Structure and Control (SSSC'98), volume 31, 1998, pp. 447–452. doi:https://doi.org/10.1016/S1474-6670(17)42032-5.

[18] P. Bouyer, S. Jaziri, N. Markey, On the determinization of timed systems, in: FORMATS, volume 10419 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 25–41.

[19] M. Krichen, S. Tripakis, Conformance testing for real-time systems, Formal Methods in System Design 34 (2009) 238–304. doi:10.1007/s10703-009-0065-1.

[20] N. Bertrand, A. Stainer, T. Jéron, M. Krichen, A game approach to determinize timed automata, Formal Methods in System Design 46 (2015) 42–80.

[21] N. Bertrand, A. Stainer, T. Jéron, M. Krichen, A game approach to determinize timed automata, in: International Conference on Foundations of Software Science and Computational Structures, Springer, Berlin, Heidelberg, 2011, pp. 245–259.

[22] L. Clemente, S. Lasota, R. Piórkowski, Determinisability of one-clock timed automata, 2020. arXiv:2007.09340.

[23] L. Fribourg, A Closed-Form Evaluation for Extended Timed Automata, Technical Report, CNRS & ECOLE NORMALE SUPERIEURE DE CACHAN, 1998.

[24] M. Bojańczyk, S. Lasota, A machine-independent characterization of timed languages, in: Proc. ICALP 2012, 2012, pp. 92–103.

[25] M. Krichen, S. Tripakis, Interesting properties of the real-time conformance relation tioco, in: K. Barkaoui, A. Cavalcanti, A. Cerone (Eds.), Theoretical Aspects of Computing - ICTAC 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 317–331.