Testing Real-Time Systems Using Determinization Techniques for Automata over Timed Domains

Moez Krichen[1,2](✉)

[1] Faculty of Computer Science and Information Technology, Al-Baha University, Al Bahah, Kingdom of Saudi Arabia
[2] ReDCAD Research Laboratory, University of Sfax, Sfax, Tunisia
moez.krichen@redcad.org

Abstract. In this work, we are interested in Model-Based Testing for Real-Time Systems. The proposed approach is based on the use of the model of Automata over Timed Domains (ATD) which corresponds to an extension of the classical Timed Automaton Model. First, we explain the main advantages of adopting this new formalism. Then, we propose a testing framework based on ATD and which is an extension of our initial framework presented in previous contributions. We extend the notion of correctness requirements (soundness and completeness) along with the notion of timed input-output conformance relation (tioco) used to compare between implementations and specifications. Moreover we propose a determinization technique used to generate test cases. Finally, several possible extensions of the present work are proposed.

Keywords: Model-Based Testing · Real-time systems · Automaton over timed domains · Correctness · Conformance relation · Determinization

1 Introduction

In general MBT (model based testing) [12] consists in describing the behavior of the SUT (system under test) using a particular adequate formalism and then generating automatically test scenarios from the considered description with respect to some coverage criteria adopting some selection methods. The following step consists in executing the obtained case studies on the SUT and collecting the corresponding verdicts in order to check whether the implementation conforms to its specification or not.

This paper extends some of our previous contributions [5–9] about MBT for real-time systems. Theses works were mainly built on the classical timed automaton model [1]. Our new proposed approach is mainly inspired by [3,4]. We adopt a new variant of timed automata called automata over timed domains (ATD). This new variant allows to model a much wider class of timed systems and it is equipped with a determinization technique which can be used for test generation.

Next in Sect. 2 we propose some definitions related to the proposed formalism. In Sect. 3 we give details about the determinization procedure for ATD. Section 4 introduces the adopted testing framework. Finally Sect. 5 proposes some directions for future work.

2 Definitions

2.1 Timed Domains and Updates

Let $\mathbb{N}$ (respectively $\mathbb{R} \geq 0$) be the set of natural numbers (respectively the set of non-negative real-numbers). *Timed domains* are introduced to represent the progression of continuous entities. A timed domain $\mathcal{D}om$ is made of a set of values denoted by $\mathcal{V}al$ and a time transition function denoted by $\rightsquigarrow$ encoding the progression of those values when time evolves.

Definition 1. *A* timed domain *is a tuple* $\mathcal{D}om = \langle \mathcal{V}al, \rightsquigarrow \rangle$ *such that:*

- $\mathcal{V}al$ *is the set of values;*
- $\rightsquigarrow : \mathcal{V}al \times \mathbb{R}_{\geq 0} \rightarrow \mathcal{V}al$ *is the time transition function such that for all* $val \in \mathcal{V}al$ *and all* $t, t' \in \mathbb{R}_{\geq 0}$ *we have* $\rightsquigarrow (\rightsquigarrow (val, t), t') = \rightsquigarrow (val, t + t')$*.*

For simplicity we will write $val \overset{t}{\rightsquigarrow} val'$ instead of $\rightsquigarrow (val, t) = val'$. Moreover we consider the particular symbol $\perp$ which is assigned to the considered resource as soon as it becomes inactive and no longer evolves over time (that is for each $t \in \mathbb{R}_{\geq 0}$ we have $\perp \overset{t}{\rightsquigarrow} \perp$).

For the timed domains $\mathcal{D}om = \langle \mathcal{V}al, \rightsquigarrow_{\mathcal{V}al} \rangle$ and $\mathcal{D}om' = \langle \mathcal{V}al', \rightsquigarrow_{\mathcal{V}al'} \rangle$ we associate the *product*

$$\mathcal{D}om \times \mathcal{D}om' = \langle \mathcal{V}al_{\mathsf{Prod}}, \rightsquigarrow_{\mathsf{Prod}} \rangle$$

such that

$$\mathcal{V}al_{\mathsf{Prod}} = (\mathcal{V}al \cup \{\perp\}) \times (\mathcal{V}al' \cup \{\perp\})$$

and for $(val, val') \in \mathcal{V}al_{\mathsf{Prod}}$ and $t \in \mathbb{R}_{\geq 0}$ we have

$$\rightsquigarrow_{\mathsf{Prod}} \big((val, val'), t\big) = \big(\rightsquigarrow_{\mathcal{V}al} (val, t), \rightsquigarrow_{\mathcal{V}al'} (val', t)\big).$$

For $n \in \mathbb{N}_{>1}$, we define the timed domain $\mathcal{D}om^n$ inductively as

$$\mathcal{D}om^1 = \mathcal{D}om$$

and

$$\mathcal{D}om^{n+1} = \mathcal{D}om \times \mathcal{D}om^n.$$

Moreover, for $\mathcal{D}om = \langle \mathcal{V}al, \rightsquigarrow_{\mathcal{V}al} \rangle$ we define the timed domain

$$\mathcal{P}(\mathcal{D}om) = \langle \mathcal{V}al_{\mathcal{P}}, \rightsquigarrow_{\mathcal{P}} \rangle$$

such that

$$\mathcal{V}al_{\mathcal{P}} = \mathcal{P}(\mathcal{V}al)$$

and

$$\forall V \in \mathcal{V}al_{\mathcal{P}} \cdot \forall t \in \mathbb{R}_{\geq 0} : \rightsquigarrow_{\mathcal{P}} (V, t) = \{\rightsquigarrow_{\mathcal{V}al} (val, t) \mid val \in V\}.$$

Definition 2. *Consider a timed domain $\mathcal{D}om = \langle \mathcal{V}al, \rightsquigarrow \rangle$ and an alphabet Δ. An* update set *for $\mathcal{D}om$ and Δ is a set $\mathcal{U} \subseteq \Delta \times \mathcal{V}al^{\mathcal{V}al}$.*

For an update set $\mathcal{U}$ and a symbol $\delta \in \Delta$, we define the set

$$\mathcal{U}_\delta = \{y \in \mathcal{V}al^{\mathcal{V}al} \mid (\delta, y) \in \mathcal{U}\}.$$

An element from $\mathcal{U}_\delta$ is called a δ-update.[1]

Let $\mathcal{D}om$ and $\mathcal{D}om'$ be two timed domains. Moreover consider two update sets $\mathcal{U}$ (for $\mathcal{D}om$ and Δ) and $\mathcal{U}'$ (for $\mathcal{D}om'$ and Δ). We then define the update set $\mathcal{U} \times \mathcal{U}'$ with respect to $\mathcal{D}om \times \mathcal{D}om'$ and Δ such that

$$\mathcal{U} \times \mathcal{U}' = \big\{\big(\delta, (y, y')\big) \mid \delta \in \Delta \ \wedge \ (y, y') \in \mathcal{U}_\delta \times \mathcal{U}'_\delta\big\}.$$

For $n \in \mathbb{N}_{>1}$, we define the update set $\mathcal{U}^n$ for $\mathcal{D}om^n$ and Δ inductively as

$$\mathcal{U}^1 = \mathcal{U}$$

and

$$\mathcal{U}^{n+1} = \mathcal{U} \times \mathcal{U}^n.$$

We also define the update set $\mathcal{P}(\mathcal{U})$ with respect to $\mathcal{P}(\mathcal{D}om)$ and Δ such that

$$\mathcal{P}(\mathcal{U}) = \big\{\big(\delta, \mathcal{Y}\big) \in \Delta \times \mathcal{P}(\mathcal{V}al)^{\mathcal{P}(\mathcal{V}al)} \mid \forall V \subseteq \mathcal{V}al \cdot \mathcal{Y}(V) = \bigcup_{(y, val) \in \mathcal{U}_\delta \times V} \{y(val)\}\big\}.$$

2.2 Automata over Timed Domains (ATD)

Definition 3. *Consider a timed domain $\mathcal{D}om = \langle \mathcal{V}al, \rightsquigarrow \rangle$ and an update set $\mathcal{U}$ for $\mathcal{D}om$ over Δ. An* automaton *on $\mathcal{D}om$ and $\mathcal{U}$ is a tuple $\mathcal{A} = \langle S, s_{ini}, v_{ini}, E, T \rangle$ where:*

- *S is a finite set of states;*
- *$s_{ini} \in S$ is the initial state;*
- *$val_{ini} \in \mathcal{V}al$ is the initial value;*
- *$E \subseteq S \times \mathcal{V}al \times \mathcal{U} \times S$ is the set of edges;*
- *$T \subseteq S$ is the set of terminal states.*

For the automaton over timed domains (*ATD*) $\mathcal{A}$ over $\mathcal{D}om$ and $\mathcal{U}$, we consider the set $C_\mathcal{A} = S \times \mathcal{V}al$ called the set of *configurations* of $\mathcal{A}$. The ATD $\mathcal{A}$ yiels a *timed labeled transition system* (TLTS)

$$\mathcal{L}_\mathcal{A} = \langle C_\mathcal{A}, c_\mathcal{A}^{ini}, \rightarrow_\mathcal{A} \rangle$$

where $c_\mathcal{A}^{ini}$ is the initial configuration of $\mathcal{A}$ such that

$$c_\mathcal{A}^{ini} = (s_{ini}, val_{ini})$$

and

$$\rightarrow_\mathcal{A} = (\xrightarrow{t}_\mathcal{A})_{t \in \mathbb{R}_{\geq 0}} \uplus (\xrightarrow{\delta, y}_\mathcal{A})_{(\delta, y) \in \mathcal{U}}$$

[1] Or simply an update.

such that

- $(s, val) \xrightarrow{t}_{\mathcal{A}} (s', val')$ if and only if $s = s'$ and $val \overset{t}{\rightsquigarrow} val'$;
- $(s, val) \xrightarrow{\delta,y}_{\mathcal{A}} (s', val')$ if and only if $(s, val, (\delta, y), s') \in E$ and $val' = y(val)$.

Similarly, the ATD $\mathcal{A}$ yiels an *observable timed labeled transition system* (OTLTS)

$$\mathcal{OL}_{\mathcal{A}} = \langle C_{\mathcal{A}}, c_{\mathcal{A}}^{ini}, \twoheadrightarrow_{\mathcal{A}} \rangle$$

where

$$\twoheadrightarrow_{\mathcal{A}} = (\overset{t}{\twoheadrightarrow}_{\mathcal{A}})_{t \in \mathbb{R}_{\geq 0}} \uplus (\overset{\delta}{\twoheadrightarrow}_{\mathcal{A}})_{\delta \in \Delta}$$

such that

- $(s, val) \overset{t}{\twoheadrightarrow}_{\mathcal{A}} (s', val')$ if and only if $(s, val) \xrightarrow{t}_{\mathcal{A}} (s', val')$;
- $(s, val) \overset{\delta}{\twoheadrightarrow}_{\mathcal{A}} (s', val')$ if and only if $\exists y \in \mathcal{V}al^{\mathcal{V}al} : (s, val) \xrightarrow{\delta,y}_{\mathcal{A}} (s', val')$.

The first type of transitions is called *timed transitions* and the second type *discrete transitions*. For $(s, val) \in C_{\mathcal{A}}$ and $\mu \in \mathbb{R}_{\geq 0} \uplus \Delta$, we write $(s, val) \overset{\mu}{\twoheadrightarrow}_{\mathcal{A}}$ if there exists $(s', val') \in C_{\mathcal{A}}$ such that $(s, val) \overset{\mu}{\twoheadrightarrow}_{\mathcal{A}} (s', val')$.

2.3 Finitely Representable ATD

A *set of guards* is a set $\mathcal{G} \subseteq \mathcal{P}(\mathcal{V}al)$. For $\delta \in \Delta$, a $\mathcal{G}$-guarded update for δ is a couple $(G, \Gamma) \in \mathcal{G} \times \mathcal{P}(\mathcal{U}_\delta)$. For $I \subseteq \mathbb{N}$, consider $\Psi_I = \{(G_i, \Gamma_i) \mid i \in I\}$ a set of $\mathcal{G}$-guarded updates for δ. Also consider $\mathcal{A} = \langle S, s_{ini}, val_{ini}, E, T \rangle$ an automaton on $\mathcal{D}om$ and $\mathcal{U}$. A pair of states (s, s') of $\mathcal{A}$ is said to be *compatible* with Ψ_I if the two following conditions hold:

- $\forall\, i \in I. \forall\, val \in G_i. \forall\, y \in \Gamma_i : (s, val, (\delta, y), s') \in E$;
- $\forall\, (s, val, (\delta, y), s') \in E. \exists\, i \in I : v \in G_i \wedge y \in \Gamma_i$.

Definition 4. *The ATD $\mathcal{A}$ is said to be* finitely representable *using $\mathcal{G}$ if for every pair of states (s, s') of $\mathcal{A}$ and for every $\delta \in \Delta$, there is a finite set Ψ of $\mathcal{G}$-guarded updates for δ, such that (s, s') is compatible with Ψ.*

2.4 Deterministic ATD (DATD)

Consider the ATD $\mathcal{A} = \langle S, s_{ini}, val_{ini}, E, T \rangle$ on $\mathcal{D}om$ and $\mathcal{U}$. Let (s, v) be a possible configuration of $\mathcal{A}$. A *timed trace* from (s, val) is a sequence $ttr = (s_i, val_i)_{0 \leq i \leq n}$ such that:

- $(s_0, val_0) = (s, val)$;
- $\forall 1 \leq i \leq n. \exists w_i \in \mathbb{R}_{\geq 0} \uplus \Delta : (s_i, val_i) \overset{w_i}{\twoheadrightarrow}_{\mathcal{A}} (s_{i+1}, val_{i+1})$.

The timed trace ttr is said to be *produced* by the *timed word* $tw = (w_i)_{1 \leq i \leq n}$. In this case we write $(s, val) \overset{tw}{\twoheadrightarrow}_{\mathcal{A}} (s_n, val_n)$ and $(s, val) \overset{tw}{\twoheadrightarrow}_{\mathcal{A}}$. The duration of tw is defined as follows

$$\mathcal{D}uration(tw) = \sum_{1 \leq i \leq n} |w_i|$$

where

$$|w_i| = \begin{cases} w_i & \text{if} w_i \in \mathbb{R}_{\geq 0} \\ 0 & \text{otherwise} \end{cases}$$

that is $\mathcal{D}uration(tw)$ denotes the complete amount of time consumed during the execution of the timed word tw.

In general given a timed word $tw \in (\mathbb{R}_{\geq 0} \uplus \Delta)^*$, the set $TTr_{\mathcal{A}}((s, val), tw)$ stands for the set of timed traces produced by tw starting from (s, val).

Definition 5. *The ATD $\mathcal{A}$ is said to be* deterministic *if for any timed word $tw \in (\mathbb{R}_{\geq 0} \uplus \Delta)^*$ the cardinality of $TTr_{\mathcal{A}}((s_{ini}, val_{ini}), tw)$ is less or equal to one.*

For a positive integer n, we say that the timed word $tw \in (\mathbb{R}_{\geq 0} \uplus \Delta)^n$ is *accepted* by the ATD $\mathcal{A}$ if there is a timed trace $ttr = (s_i, val_i)_{0 \leq i \leq n}$ such that $ttr \in TTr_{\mathcal{A}}((s_{ini}, val_{ini}), tw)$ and $s_{n+1} \in T$ (i.e. s_{n+1} is a terminal state of $\mathcal{A}$). In this case all the configurations (s_i, val_i) are said to be *reachable*. The set of accepted timed words by $\mathcal{A}$ is denoted $\mathcal{L}ang(\mathcal{A})$ and the set of reachable configurations is denoted $\mathcal{R}each(\mathcal{A})$.

3 Determinization of Non-Deterministic ATD (NDATD)

Consider the (possibly) non-deterministic ATD $\mathcal{A} = \langle S, s_{ini}, val_{ini}, E, T \rangle$ on $\mathcal{D}$ and $\mathcal{U}$ and let Δ be the alphabet corresponding to the update set $\mathcal{U}$. For simplicity we assume that

$$S = \{s_1, \cdots, s_p\}$$

such that $p \in \mathbb{N}_{>0}$ and $s_{ini} = s_1$. For every state $s \in S$, we let $\mathsf{index}(s)$ denote the index of s. That is if $s = s_i$ then $\mathsf{index}(s) = i$. For each $1 \leq i \leq p$, we consider the set $Val_i \subseteq \mathcal{V}al$ which corresponds to the set of values corresponding to the state s_i.

For $\mathbf{V} = (V_i)_{1 \leq i \leq p} \in \mathcal{P}(\mathcal{V}al)^p$ we consider the set

$$S_{\mathbf{V}} = \{s \in S \mid V_{\mathsf{index}(s)} \neq \emptyset\}$$

which is the group of states the system may occupy when the values for the different states are given by $\mathbf{V}$.

For $\delta \in \Delta$ and $1 \leq i \leq j \leq p$, we associate the set

$$\lambda_\delta^{i \to j} = \{(val, y) \in \mathcal{V}al \times \mathcal{U}_\delta \mid (s_i, val, (\delta, y), s_j) \in E\}$$

which corresponds the different ways allowing to move from state s_i to state s_j. We also define

$$\lambda_\delta^{\to j} = (\lambda_\delta^{i \to j})_{1 \leq i \leq p}$$

which in turn records all the ways which allow to reach state s_j starting from any other state of the ATD $\mathcal{A}$.

For the considered letter δ and each $\lambda_\delta^{i\to j}$, we associate the *successor* function

$$\mathsf{succ}_{\delta,\lambda_\delta^{i\to j}} : \mathcal{V}al \to \mathcal{P}(\mathcal{V}al)$$

such that for $v \in \mathcal{V}al$ we have

$$\mathsf{succ}_{\delta,\lambda_\delta^{i\to j}}(val) = \{y(val) \mid (val, y) \in \lambda_\delta^{i\to j}\}$$

which collects all possible obtained values of *val* after executing instructions in $\lambda_\delta^{i\to j}$.

In a natural way we extend $\mathsf{succ}_{\delta,\lambda_\delta^{i\to j}}$ to elements from $\mathcal{P}(\mathcal{V})al$ and we define the function

$$\mathsf{Succ}_{\delta,\lambda_\delta^{i\to j}} : \mathcal{P}(\mathcal{V}al) \to \mathcal{P}(\mathcal{V}al)$$

such that for $V \subseteq \mathcal{V}al$ we have

$$\mathsf{Succ}_{\delta,\lambda_\delta^{i\to j}}(V) = \bigcup_{val \in V} \mathsf{succ}_{\delta,\lambda_\delta^{i\to j}}(val)$$

which this time collects the possible obtained values of all elements in V after executing instructions in $\lambda_\delta^{i\to j}$.

Moreover we define the function

$$\mathsf{Succ}_{\delta,\lambda_\delta^{\to j}} : \mathcal{P}(\mathcal{V}al)^p \to \mathcal{P}(\mathcal{V}al)$$

such that for $\mathbf{V} = (V_i)_{1\le i\le p} \in \mathcal{P}(\mathcal{V}al)^p$ we have

$$\mathsf{Succ}_{\delta,\lambda_\delta^{\to j}}(\mathbf{V}) = \bigcup_{1\le i\le p} \mathsf{Succ}_{\delta,\lambda_\delta^{i\to j}}(V_i)$$

which aggregates the possible updated values of $V_1, \cdots, V_p$ following respectively the instructions in $\delta^{1\to j}, \cdots, \delta^{p\to j}$.

Furthermore, we define the function

$$\mathsf{Succ}_{\delta,\mathcal{A}} : \mathcal{P}(\mathcal{V}al)^p \to \mathcal{P}(\mathcal{V}al)^p$$

such that for $\mathbf{V}al \in \mathcal{P}(\mathcal{V}al)^p$ we have

$$\mathsf{Succ}_{\delta,\mathcal{A}}(\mathbf{V}) = \left(\mathsf{Succ}_{\delta,\lambda_\delta^{\to 1}}(\mathbf{V}), \cdots, \mathsf{Succ}_{\delta,\lambda_\delta^{\to p}}(\mathbf{V})\right)$$

which can be seen as the successor of $\mathbf{V}$ after the execution of δ.

Lemma 1. *Let $\mathbf{V} = (V_i)_{1\le i\le p} \in \mathcal{P}(\mathcal{V}al)^p$ and $\mathbf{V}' = (V_i')_{1\le i\le p} = \mathsf{Succ}_{\delta,\mathcal{A}}(\mathbf{V})$. Then for every $s' \in S$ and $v' \in \mathcal{V}$:*

$$val' \in V'_{\mathsf{index}(s')} \Leftrightarrow \exists (s, val, (\delta, y), s') \in E \text{ s.t. } v \in V_{\mathsf{index}(s)} \text{ and } v' = y(val).$$

Finally we define the update set $\mathcal{P}^p(\mathcal{U})$ with respect to $\mathcal{P}(\mathcal{D})^p$ and Δ such that[2]

$$\mathcal{P}^p(\mathcal{U}) = \{(\delta, \mathsf{Succ}_{\delta,\mathcal{A}}) \mid \delta \in \Delta\}.$$

We now have all the ingredients to define a deterministic ATD (*DATD*)

$$\mathcal{A}_{det} = \langle S_{det}, s_{ini}^{det}, val_{ini}^{det}, E_{det}, T_{det}\rangle$$

on $\mathcal{P}(\mathcal{D})^p$ and $\mathcal{P}^p(\mathcal{U})$ which is equivalent to the considered NDATD $\mathcal{A}$. The proposed DATD $\mathcal{A}_{det}$ is defined as follows:

- $S_{det} = \mathcal{P}(S)$;
- $s_{ini}^{det} = \{s_{ini}\}$;
- $val_{ini}^{det} = \big(\{val_{ini}\}, \emptyset, \cdots, \emptyset\big) \in \mathcal{P}(\mathcal{V})^p$;
- E_{det} is to the set of transitions $(S_{\mathbf{V}}, \mathbf{V}, (\delta, \mathsf{Succ}_{\delta,\mathcal{A}}), S')$ such that $\mathbf{V} \in \mathcal{P}(\mathcal{V}al)^p$ and $S' = S_{\mathbf{V}'}$ with $\mathbf{V}' = \mathsf{Succ}_{\delta,\mathcal{A}}(\mathbf{V})$;
- $T_{det} = \{S' \subseteq S \mid S' \cap T \neq \emptyset\}$.

4 Testing Framework

4.1 ATD with Inputs and Outputs (ATDIO)

Consider the ATD $\mathcal{A} = \langle S, s_{ini}, v_{ini}, E, T\rangle$ on $\mathcal{D}$ and $\mathcal{U}$ and let Δ be the alphabet corresponding to the update set $\mathcal{U}$. We assume that the alphabet Δ is split into two disjoint sets namely Δ_I a set of inputs and Δ_O a set of outputs (i.e., $\Delta = \Delta_I \sqcup \Delta_O$)[3]. In this case the ATD $\mathcal{A}$ is called an *ATD with inputs and outputs* (ATDIO). Moreover for we suppose that all the states of $\mathcal{A}$ are terminal (i.e., $T = E$).

The ATDIO $\mathcal{A}$ is said to be *input-enabled* if for any reachable configuration $conf \in \mathcal{R}each(\mathcal{A})$ and any input symbol $inp \in \Delta_I$ we have $conf \xrightarrow{inp}\!\!\!\!\rightarrow_{\mathcal{A}}$.

Moreover the considered ATDIO is called *non-blocking* if for any reachable configuration $conf \in \mathcal{R}each(\mathcal{A})$ and any duration $t \in \mathbb{R}_{\geq 0}$ there exists $tw \in (\mathbb{R}_{\geq 0} \uplus \Delta_O)^*$ such that $conf \xrightarrow{tw}\!\!\!\!\rightarrow_{\mathcal{A}}$ and $\mathcal{D}uration(tw) = t$.

Next we suppose that the specification of the system under test and the implementation are given as two non-blocking ATDIO $\mathcal{S}p$ and $\mathcal{I}m$ respectively.[4]

4.2 Parallel Composition of OTLTS with Inputs and Outputs

Given two OTLTS with inputs and outputs LTS_1 and LTS_2, we define the parallel product $LTS_1||LTS_2$. For $i = 1, 2$, $LTS_i = (Q_i, q_0^i, \Delta_{\mathsf{I}}^i \cup \Delta_{(3-i)\rightarrow i}, \Delta_{\mathsf{O}}^i \cup \Delta_{i\rightarrow(3-i)}, T_d^i, T_t^i)$. The sets Δ_{I}^1, Δ_{O}^1, Δ_{I}^2, Δ_{O}^2, $\Delta_{1\rightarrow 2}$ and $\Delta_{2\rightarrow 1}$ are pairwise disjoint. The two OTLTS synchronize on shared common actions $\Delta_{1\leftrightarrow 2} = \Delta_{1\rightarrow 2} \cup \Delta_{2\rightarrow 1}$ and time delays. The parallel product of the two OTLTS is

$$LTS_1||LTS_2 = (Q, (q_0^1, q_0^2), \Delta_{\mathsf{I}}, \Delta_{\mathsf{O}}, T_d, T_t)$$

[2] Note that $\mathcal{P}^p(\mathcal{U})$ is not the same as $\mathcal{P}(\mathcal{U})^p$.

[3] $\sqcup$ stands for the disjoint union symbol.

[4] We do not assume $\mathcal{I}m$ is known.

such that

$$\Delta_\mathsf{I} = \bigcup_{i=1,2} \Delta_\mathsf{I}^i,\ \Delta_\mathsf{O} = \bigcup_{i=1,2} \Delta_\mathsf{O}^i$$

and Q, T_d and T_t are the smallest groups of elements such that:

- $(q_0^1, q_0^2) \in Q$;
- For $(q_1, q_2) \in Q$ and $\delta \in \mathsf{R}$:

$$q_1 < aq_1' \in T_d^1 \Rightarrow (q_1', q_2) \in S \wedge (q_1, q_2) < a(q_1', q_2) \in T_d;$$

- For $(q_1, q_2) \in Q$ and $a \in \Delta_\mathsf{I}^1 \cup \Delta_\mathsf{O}^1 \cup \{\tau_1\}$:

$$q_1 < aq_1' \in T_d^1 \Rightarrow (q_1', q_2) \in S \wedge (q_1, q_2) < a(q_1', q_2) \in T_d;$$

- For $(q_1, q_2) \in Q$ and $a \in \Delta_\mathsf{I}^2 \cup \Delta_\mathsf{O}^2 \cup \{\tau_2\}$:

$$q_2 < aq_2' \in T_d^2 \Rightarrow (q_1, q_2') \in S \wedge (q_1, q_2) < a(q_1, q_2') \in T_d;$$

- For $(q_1, q_2) \in Q$ and $a \in \Delta_{1\leftrightarrow 2}$:

$$q_1 < aq_1' \in T_d^1 \wedge q_2 < aq_2' \in T_d^2 \Rightarrow (q_1', q_2') \in Q \wedge (q_1, q_2) < \tau_a(q_1', q_2') \in T_d.$$

4.3 Conformance Relation

Given a ATDIO $\mathcal{A}$ and a timed word $tw \in (\mathbb{R}_{\geq 0} \uplus \Delta)^*$, $\mathcal{A}$ after tw is the set of configurations of $\mathcal{A}$ that can be reached after the execution of tw. Formally:

$$\mathcal{A}\ \mathsf{after}\ tw = \{conf \in C_\mathcal{A} \mid c_{ini}^\mathcal{A} \xrightarrow{tw}\!\!\!\!\rightarrow_\mathcal{A} conf\}.$$

Given configuration $conf \in C_\mathcal{A}$, $\mathsf{out}(conf)$ is the set of all observations (either outputs or the elapsing of time) that may happen when the system is at configuration $conf$. The definition is extended in a natural way to a set of configurations $Conf$. Formally:

$$\mathsf{out}(conf) = \{\mu \in \mathbb{R}_{\geq 0} \uplus \Delta_O \mid conf \xrightarrow{\mu}\!\!\!\!\rightarrow_\mathcal{A}\ \}\ ,\ \mathsf{out}(Conf) = \bigcup_{conf \in Conf} \mathsf{out}(conf).$$

The definition of the relation tioco [10,11] is as follows:

$$\mathcal{I}m\ \mathsf{tioco}\ \mathcal{S}p\ \ \textit{iff}\ \ \forall tw \in \mathcal{L}ang(\mathcal{S}p)\ :\ \mathsf{out}(\mathcal{I}m\ \mathsf{after}\ tw) \subseteq \mathsf{out}(\mathcal{S}p\ \mathsf{after}\ tw).$$

The relation indicates that the implementation $\mathcal{I}m$ conforms to the specification $\mathcal{S}p$ if and only if for any timed word tw of $\mathcal{S}p$, the set of outputs (including time elapse) of $\mathcal{I}m$ after the execution of tw is a subset of the set of outputs that can be generated by $\mathcal{S}p$.

4.4 Timed Test Cases

A timed test case for the specification $\mathcal{S}p$ over Δ_τ is a total function

$$TTest : (\mathbb{R}_{\geq 0} \uplus \Delta)^* \to \varDelta_{\mathsf{I}} \cup \{\mathsf{WT}, \mathsf{PS}, \mathsf{FL}\}.$$

$TTest(tw)$ indicates the action that must executed by the tester once it observes tw. If $TTest(tw) = inp \in \varDelta_{\mathsf{I}}$ then the tester produces input inp. If $TTest(tw) = \mathsf{WT}$ then the tester lets time elapse (waits). If $TTest(tw) \in \{\mathsf{PS}, \mathsf{FL}\}$ then the tester emits a verdict and stops.

The execution of $TTest$ on $\mathcal{I}m$ may be seen as the *parallel composition* of the OTLTS with inputs and outputs defined by $TTest$ and $\mathcal{I}m$. The product TIOLTS is denoted by $\mathcal{I}m \| TTest$. In a formal fashion, we announce that the implementation $\mathcal{I}m$ *passes* $TTest$, denoted $\mathcal{I}m$ passes $TTest$, if state FL can not be reached in $\mathcal{I}m \| TTest$. We declare that the implementation passes (respectively fails) the test suite $\mathcal{TT}$ if it passes all tests (respectively fails at least one test) in $\mathcal{TT}$. $\mathcal{TT}$ is said to be *sound with respect to* $\mathcal{S}p$ if

$$\forall \mathcal{I}m \ : \ \mathcal{I}m \ \mathsf{tioco}\ \mathcal{S}p \Rightarrow \mathcal{I}m \ \mathsf{passes}\ \mathcal{TT}.$$

Similarly $\mathcal{TT}$ is said to be *complete with respect to* $\mathcal{S}p$ if

$$\forall \mathcal{I}m \ : \ \mathcal{I}m \ \mathsf{passes}\ \mathcal{TT} \Rightarrow \mathcal{I}m \ \mathsf{tioco}\ \mathcal{S}p.$$

Our goal it then to produce test suites which are both sound and complete. More precisely, we aim to generate timed tests in the form of deterministic ATDIO and which are finitely representable. For that purpose we need to use the determinization technique proposed in Sect. 3.

5 Future Work

The work proposed in this paper is at its beginning. In the future we need to work on many aspects:

- First, we need to extend the presented framework to the case where the specification of the SUT is given as a product of ATDIO and not simply one ATDIO. In this way we can deal with distributed and multi-components systems.
- Second, we should find a way which guarantees that the generated timed tests are finitely representable so that we can store them and execute them later on. For that, we need to use some approximation techniques based on game theory techniques like the ones proposed in [2].
- Third, we need to use some coverage and selection techniques which allow to reduce the size of generated test cases and to efficiently deal the state explosion problem usually encountered when following model-based approaches.

References

1. Alur, R., Dill, D.: The theory of timed automata. In: de Bakker, J.W., Huizing, C., de Roever, W.P., Rozenberg, G. (eds.) REX 1991. LNCS, vol. 600, pp. 45–73. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0031987
2. Bertrand, N., Stainer, A., Jéron, T., Krichen, M.: A game approach to determinize timed automata. Formal Methods Syst. Des. **46**(1), 42–80 (2015)
3. Bojańczyk, M., Lasota, S.: A machine-independent characterization of timed languages. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012. LNCS, vol. 7392, pp. 92–103. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31585-5_12
4. Bouyer, P., Jaziri, S., Markey, N.: On the determinization of timed systems. In: Abate, A., Geeraerts, G. (eds.) FORMATS 2017. LNCS, vol. 10419, pp. 25–41. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65765-3_2
5. Krichen, M.: A formal framework for conformance testing of distributed real-time systems. In: Lu, C., Masuzawa, T., Mosbah, M. (eds.) OPODIS 2010. LNCS, vol. 6490, pp. 139–142. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17653-1_12
6. Krichen, M.: A formal framework for black-box conformance testing of distributed real-time systems. Int. J. Crit. Comput. Based Syst. **3**(1/2), 26–43 (2012). https://doi.org/10.1504/IJCCBS.2012.045075. http://dx.doi.org/10.1504/IJCCBS.2012.045075
7. Krichen, M., Tripakis, S.: Black-box conformance testing for real-time systems. In: Graf, S., Mounier, L. (eds.) SPIN 2004. LNCS, vol. 2989, pp. 109–126. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24732-6_8
8. Krichen, M., Tripakis, S.: Real-time testing with timed automata testers and coverage criteria. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 134–151. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_11
9. Krichen, M., Tripakis, S.: An expressive and implementable formal framework for testing real-time systems. In: Khendek, F., Dssouli, R. (eds.) TestCom 2005. LNCS, vol. 3502, pp. 209–225. Springer, Heidelberg (2005). https://doi.org/10.1007/11430230_15
10. Krichen, M., Tripakis, S.: Interesting properties of the real-time conformance relation tioco. In: Barkaoui, K., Cavalcanti, A., Cerone, A. (eds.) ICTAC 2006. LNCS, vol. 4281, pp. 317–331. Springer, Heidelberg (2006). https://doi.org/10.1007/11921240_22
11. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Formal Methods Syst. Des. **34**(3), 238–304 (2009). https://doi.org/10.1007/s10703-009-0065-1
12. Tretmans, J.: Testing concurrent systems: a formal approach. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 46–65. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48320-9_6. http://dl.acm.org/citation.cfm?id=646734.701460