Mutant Equivalence as Monotonicity in Parametric Timed Games

Davide Basile*†, Maurice H. ter Beek†, Hendrik Göttmann‡ and Malte Lochau§
†ISTI–CNR, Pisa, Italy
Email: {davide.basile,maurice.terbeek}@isti.cnr.it
‡TU Darmstadt, Germany
Email: hendrik.goettmann@es.tu-darmstadt.de
§University of Siegen, Germany
Email: malte.lochau@uni-siegen.de

***Abstract*—The detection of faults in software systems can be enhanced effectively by model-based mutation testing. The efficiency of this technique is hindered when mutants are equivalent to the original system model, since this makes them useless. Recently, the application of model-based mutation testing to real-time systems modelled as timed games has been investigated, which has resulted in guidelines for statically avoiding equivalent mutants.**

In this paper, we recast this problem into the framework of parametric timed games. We then prove a correspondence between theoretical results for the detection of equivalent mutants in timed games and the property of monotonicity that is known to hold for a sub-class of parametric timed games called L/U parametric timed games. The presented results not only simplify the theory underlying the detection of equivalent mutants in timed games, but at the same time they improve the expressiveness of a known decidable fragment of parametric timed games for which monotonicity holds.

***Index Terms*—parametric timed games, mutant equivalence, model-based mutation testing**

I. Introduction

Model-based testing methods automate the generation of test cases by using formal models of the system [1]. Moreover, they help to manage the time-sensitiveness of requirements, which makes testing a difficult problem. Testing a formal model rather than source code allows to detect, among others, misinterpretations of requirements or issues arising from time-dependent interactions of the system with its environment, which would be harder to detect at source code level. A suite of tests generated from a model can be translated into a set of tests for a corresponding implementation.

To evaluate the thoroughness of test cases or to support their generation [2], [3], one commonly uses model-based mutation testing [4], [5]. To this aim, *mutation* operators that simulate possible faults in the system are applied to the model, resulting in a so-called *mutant*. Given a set of mutants, the effectiveness of a set of test cases is evaluated according to the number of mutants that it detects (i.e., mutants producing different output than the original system). Test cases generated from a mutant are capable of detecting bugs mimicked by that mutation, and generally require to test corner cases.

* Corresponding author, first author

It has been shown in the literature [2] that mutation-based testing is more effective in finding real faults than other techniques [3], [6], [7].

Typically, a large number of mutations is required in order to build effective test cases. In model-based mutation testing, random applications of mutation operators may lead to many useless mutations since they generate mutants that may exhibit the same observable external behaviour as the original system, thus leading to a scalability problem. Concerning model-based mutation testing, the *equivalent mutant* problem generalizes to that of detecting *subsumed* mutants, which have less (or equal) behaviour than the original system model (i.e., from a tester's point of view, the subsumed mutant is not distinguishable from the original system). In such cases, no test case can be generated to differentiate the mutant from the original system, leading to useless analyses and a waste of computational resources.

The problem of detecting equivalent mutants in (model-based) mutation testing has recently been investigated in [4], [8]–[12]. In several of these papers, the real-time systems are modelled as timed games. In particular, in [11], [12] several formal results identify specific conditions under which, by construction, given mutations provide subsumed mutants, from which a set of guidelines (called commandments) for avoiding subsumed mutants was synthesised.

In this paper, we show a connection between the problem of detecting equivalent mutants in mutation testing of real-time models and the monotonicity property of a fragment of parametric timed games. We adopt a symbolic approach to the problem of detecting subsumed mutants in timed games, by using the framework of parametric timed games.

The results presented in this paper contribute to research on model-based mutation testing as well as to research on parametric timed games. In particular, our contributions to these two fields are as follows:

1) Concerning model-based mutation testing, we prove a correspondence between the theoretical results to detect equivalent mutants in timed games as presented in [11], [12] and the property of monotonicity of a fragment of parametric timed games called L/U parametric timed games. Due to the parametric setting in which we operate,

this correspondence simplifies the theoretical results from [11], [12]. In fact, when restricted to L/U parametric timed games, the monotonicity property alone is responsible for over half of the commandments listed in [12];

2) Concerning parametric timed games, we extend the fragment of L/U parametric timed games originally introduced in [13] to include parametric constraints also on invariants. This is necessary to prove that the above correspondence also holds for commandments that predicate on mutations of invariants, which are in fact imported into this fragment. We prove that monotonicity holds for this specific fragment, thus extending and improving the expressiveness of a known decidable fragment of parametric timed games. This fragment is now capable of expressing parametric invariants, a crucial ingredient for timed specifications [14] (e.g., without invariants it is not possible to specify that an event must occur at a precise instant of time, because it is the invariants that provide control over time).

As a further contribution, the presented results pave the way to the usage of tools developed in the area of parametric timed games, such as [15], [16], for the problem of model-based mutation testing, and we will identify some future research goals in this direction.

A. Structure of the Paper

We start with a discussion of related work in Section II. We provide some background on timed games and parametric timed games in Section III. The extension of the fragment of L/U parametric timed games and the monotonicity proof are presented in Section IV. The correspondence between the monotonicity property and the violation of the commandments from [11], [12] is presented in Section V. Finally, conclusions and future work are provided in Section VI.

II. Related Work

Guidelines or commandments for statically detecting subsumed mutants have originally been introduced in [11], [12], together with a proof-of-concept implementation applied to various case studies, showing an improvement in mutant generation with respect to randomly applying mutations. In [11], [12], a theoretical result on timed games refinement is associated to each commandment. We, instead, propose a compact theory that expresses many of these results as instances of a single monotonicity property of a duly extended fragment of L/U parametric timed games, by exploiting the symbolic reasoning of parametric timed games.

The subclass called L/U parametric timed games is discussed in [13], [17] as a decidable fragment of the class of parametric timed games for the synthesis of reachability properties for safety games. The proof of decidability relies upon the property of monotonicity of L/U parametric timed games. Here we extend the fragment of L/U parametric timed games in [13] to include also parametric constraints on invariants while, previously, parametric constraints were only allowed on guards. We prove that the monotonicity property holds for this extended fragment, thus retaining decidability of strategy synthesis [17].

The experiments conducted in [11], [12] have been validated by using as control the refinement checking provided by Uppaal TIGA [18]. This refinement checking is internally implemented using the (non-parametric) strategy synthesis algorithm in [14]. An algorithm for strategy synthesis of parametric timed games has recently been studied in [19] and it has been implemented in the tool Romeo [15], which uses Timed Petri Nets models [20]. The results we present here could be exploited, by using a suitable extension of the above mentioned techniques and tools, to perform refinement checking of parametric timed games.

In [21], so-called configurable parametric timed automata [22] are used to generate a test-suite with full coverage using min/max delays parameters. Configurable parametric timed automata extend parametric timed automata with configurations. In [11], [12], also the mutations are organised into a product line of mutants, called featured mutant model [8]. We do not consider the extension to a product line of mutations here, but leave this for future work (cf. Section VI).

We exploit monotonicity of L/U parametric timed games to detect subsumed mutants. In [23], the converse approach is used: given a timed automaton failing some tests, mutations are introduced to search for mutants that fix the model. Similar to our approach, an initial timed automaton is abstracted into a parametric timed automaton to allow symbolic reasoning, using decidability results on problems for parametric timed automata introduced in [24]. A generic algorithm for program repair using mutations was published at FormalISE 2017 [25].

At last year's FormaliSE 2022 [26], an algorithm for generating test-suites satisfying Boolean coverage criteria was presented, in which no mutation testing was used to evaluate the generated test-suites. A methodology using mutation analysis to quantify the strength of software contracts was presented at FormaliSE 2021 [27], but the problem of equivalent mutants was not addressed.

III. Background

In this section, we recall the notions of timed games and parametric timed games.

A. Timed Games

Timed games are transition systems which can remain in a certain location only for a specific amount of time, can execute a transition only within a certain time interval, and distinguish between controllable and uncontrollable actions. Timed games are based on timed (game) automata [28], [29].

In reactive systems, one usually distinguishes between uncontrollable and controllable actions, which traditionally are assigned to inputs and outputs, respectively, whenever the environment is uncontrollable and vice versa otherwise. Concerning model-based testing, the point of view of the tester is considered. The inputs to the system are controllable whilst the outputs from the system are uncontrollable.

Time is represented by clocks whose values evolve continuously. Clocks can be regarded as chronometers: their value can

be inspected and reset, but not modified arbitrarily. Conditions over clock values are called *clock constraints*.

Definition 1 (Clock constraint): Let C be a set of clocks on a clock value domain Δ. Then the set Φ_C of *clock constraints* φ on C is inductively defined as:

$$\varphi ::= \top \mid c \sim n \mid \varphi \wedge \varphi \mid \neg\varphi$$

where $c \in C$, $\sim \in \{<, \leq, \geq, >\}$, and $n \in \Delta$.

We denote by $[\![\varphi]\!]$ the set of clock valuations that satisfy clock constraint φ, i.e., the set of total functions $v : C \to \Delta$ that assign a value to every clock. In timed games, a clock constraint can label either a location or a transition. In case it labels a location, the constraint is a location *invariant*, which defines the interval of time in which the system can remain in the location. In case it labels a transition, it is a transition *guard* specifying the interval of time during which the system can execute the transition. Note that the domain of the numeric constants in clock constraints is limited to the natural numbers. Without loss of generality, we could use real numbers. However, natural numbers facilitate the implementation of clock constraints by allowing efficient data structures.

Definition 2 (Timed game): A *timed game* is a sextuple $(L, \ell_0, \Sigma, C, I, E)$, where

- L is a set of *locations*,
- $\ell_0 \in L$ is the *initial location*,
- Σ is a set of *actions*, partitioned into *controllable* actions Σ^c and *uncontrollable* actions Σ^u,
- C is a set of *clocks*,
- $I : L \to \Phi_C$ assigns *location invariants*, and
- $E \subseteq L \times \Phi_C \times \Sigma \times 2^C \times L$ is a set of *transitions*.

Let *TG* denoted the set of timed games.

For a transition $t = (\ell, \varphi, \alpha, R, \ell')$, ℓ is the starting location, φ is the transition guard (clock constraint), α is the action triggering the transition, R is the subset of clocks to reset, and ℓ' is the target location. We may also write t as $\ell \xrightarrow{\varphi,\alpha,R} \ell'$ and omit φ and/or R when immaterial, and instead of $\{x\}$ for a reset of clock x, we may also write $x := 0$ and use the transitive closure $\to^*$.

Example 1: Figure 1 shows an example of a timed game *tg* modeling a basic tea machine, which accepts a euro coin € as input and waits at most four time units before providing tea 🍵 as output. Controllable transitions are depicted by solid lines while uncontrollable transitions are depicted by dotted lines.

Formally, the semantics of a timed game is commonly defined as an infinite transition system, whose states consist of a location and a valuation of the clocks, and transitions are alternating between two types. *Delay transitions* do not change the location of the system, but only represent the passing of time. They may occur only if the invariant of the current location is still satisfied after the delay modelled by the transition. *Discrete transitions*, instead, occur when the system moves from one location to another without passage of time. They may occur only if the current clock values satisfy both the guard of the executed transition and the invariant of the target location. After the execution of such transitions, clock values can be reset.

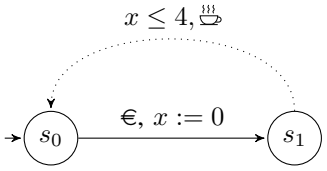

Fig. 1. A timed game *tg*

Definition 3 (Timed games semantics): The semantics of a timed game $tg = (L, \ell_0, \Sigma, C, I, E)$ is defined as the set of traces $[\![tg]\!] = \{\, w \mid w \in (\Sigma \cup \Delta)^*, (\ell_0, v_0) \xrightarrow{w}{}^* (\ell', v') \,\}$ of the transition system $(Loc \times [\![\varphi]\!], (\ell_0, v_0), \Sigma \cup \Delta, \Phi_C, E')$, where $v_0 = \{\, v(c) = 0 \mid c \in C \,\}$.

The notion of timed refinement is based on [30] and it coincides with the timed input-output conformance for input-enabled models [9], [31], where inputs (outputs, respectively) are controllable (uncontrollable, respectively).

Informally, a timed game tg_1 is a refinement of a timed game tg_2 (denoted as $tg_1 \preceq tg_2$) when the refined model tg_1 is able to mimic all controllable transitions of the original system model tg_2, while tg_2 is able to mimic all uncontrollable transitions and delays of tg_1.

Definition 4 (Timed games refinement): Let $tg_1 = (L_1, \ell_{0_1}, \Sigma_1, C_1, I_1, E_1)$ and $tg_2 = (L_2, \ell_{0_2}, \Sigma_2, C_2, I_2, E_2)$ be timed games. Then tg_1 is a refinement of tg_2, denoted as $tg_1 \preceq tg_2$, with $[\![\varphi_i]\!] : C_i \mapsto \Delta$, if there exists a binary relation $R \subseteq (L_1, [\![\varphi_1]\!]) \times (L_2, [\![\varphi_2]\!])$ that contains $s = ((\ell_{01}, v_{01}), (\ell_{02}, v_{02}))$ and which is such that for each pair of locations $((\ell_1, v_1), (\ell_2, v_2)) \in R$, the following holds:

- whenever $(\ell_2, v_2) \xrightarrow{\alpha} (\ell_2', v_2)$ for some ℓ_2' and $\alpha \in \Sigma_2^c$, then $(\ell_1, v_1) \xrightarrow{\alpha} (\ell_1', v_1)$ for some ℓ_1', $\alpha \in \Sigma_1^c$, and $((\ell_1', v_1), (\ell_2', v_2)) \in R$
- whenever $(\ell_1, v_1) \xrightarrow{\alpha} (\ell_1', v_1)$ for some ℓ_1' and $\alpha \in \Sigma_1^u$, then $(\ell_2, v_2) \xrightarrow{\alpha} (\ell_2', v_2)$ for some ℓ_2', $\alpha \in \Sigma_2^u$, and $((\ell_1', v_1), (\ell_2', v_2)) \in R$
- whenever $(\ell_1, v_1) \xrightarrow{\delta} (\ell_1, v_1')$ for some v_1' and $\delta \in \mathbb{R}_{\geq 0}$, then $(\ell_2, v_2) \xrightarrow{\delta} (\ell_2, v_2')$ for some v_2', and $((\ell_1, v_1'), (\ell_2, v_2')) \in R$

The proposed notion of refinement is slightly different from the standard one. Indeed, in Definition 4 the two systems are not forced to share the same alphabet. We show that this new definition of refinement is still a preorder.

Lemma 1: Timed games refinement is a preorder.

Proof: Reflexivity of $\preceq$ is trivial. Concerning transitivity, assume by hypothesis that $tg_1 \preceq tg_2$ and $tg_2 \preceq tg_3$ for three timed games tg_1, tg_2, and tg_3 and let R_1 and R_2 be the relations proving, respectively, that $tg_1 \preceq tg_2$ and $tg_2 \preceq tg_3$.

We prove that $R = \{\, ((\ell_1, v_1), (\ell_3, v_3)) \mid ((\ell_1, v_1), (\ell_2, v_2)) \in R_1,\ ((\ell_2, v_2), (\ell_3, v_3)) \in R_2,\ (\ell_2, v_2) \in Loc_{tg_2} \times [\![\varphi_{tg_2}]\!] \,\}$ is a relation that shows that $tg_1 \preceq tg_3$.

First, by construction, $((\ell_{0_{tg_1}}, v_{0_{tg_1}}), (\ell_{0_{tg_3}}, v_{0_{tg_3}})) \in R$. Then, for all $((\ell_1, v_1), (\ell_3, v_3)) \in R$, by construction we have that for some $(\ell_2, v_2) \in Loc_{tg_2} \times [\![\varphi_{tg_2}]\!]$, it holds that $((\ell_1, v_1), (\ell_2, v_2)) \in R_1$ and $((\ell_2, v_2), (\ell_3, v_3)) \in R_2$. Moreover:

- Whenever $(\ell_3, v_3) \xrightarrow{\alpha} (\ell'_3, v_3)$, with $\alpha \in \Sigma^c_{tg_3}$, by $tg_2 \preceq tg_3$, it follows that $(\ell_2, v_2) \xrightarrow{\alpha} (\ell'_2, v_2)$, with $\alpha \in \Sigma^c_{tg_2}$, and $((\ell'_2, v_2), (\ell'_3, v_3)) \in R_2$. By $tg_1 \preceq tg_2$, it follows that $(\ell_1, v_1) \xrightarrow{\alpha} (\ell'_1, v_1)$, with $\alpha \in \Sigma^c_{tg_1}$ and $((\ell'_1, v_1), (\ell'_2, v_2)) \in R_1$. By construction, $((\ell'_1, v_1), (\ell'_3, v_3)) \in R$.
- Whenever $(\ell_1, v_1) \xrightarrow{\alpha} (\ell'_1, v_1)$, with $\alpha \in \Sigma^u_{tg_1}$, by $tg_1 \preceq tg_2$, it follows that $(\ell_2, v_2) \xrightarrow{\alpha} (\ell'_2, v_2)$, with $\alpha \in \Sigma^u_{tg_2}$ and $((\ell'_1, v_1), (\ell'_2, v_2)) \in R_1$. By $tg_2 \preceq tg_3$, it follows that $(\ell_3, v_3) \xrightarrow{\alpha} (\ell'_3, v_3)$, with $\alpha \in \Sigma^u_{tg_3}$ and $((\ell'_2, v_2), (\ell'_3, v_3)) \in R_2$. By construction, $((\ell'_1, v_1), (\ell'_3, v_3)) \in R$.
- Whenever $(\ell_1, v_1) \xrightarrow{\delta} (\ell_1, v'_1)$, with $\delta \in \mathbb{R}_{\geq 0}$, by $tg_1 \preceq tg_2$, it follows that $(\ell_2, v_2) \xrightarrow{\delta} (\ell_2, v'_2)$ and $((\ell_1, v'_1), (\ell_2, v'_2)) \in R_1$. By $tg_2 \preceq tg_3$, it follows that $(\ell_3, v_3) \xrightarrow{\delta} (\ell_3, v_{3'})$ and $((\ell_2, v'_2), (\ell_3, v'_3)) \in R_2$. By construction, $((\ell_1, v'_1), (\ell_3, v'_3)) \in R$. ■

Refinement checking is solved as a two-player alternating timed game, where one player (playing the "whenever" transitions of Definition 4) wins if it proves that timed game tg_1 is not a refinement of timed game tg_2 and the other player (playing the "then" transitions of Definition 4) wins if it proves that tg_1 is a refinement of tg_2 [14]. If there exists a non-empty strategy for the "whenever" player to reach a winning state, then we know that tg_1 is not a refinement of tg_2, where a winning state for the "whenever" player is a state where the "then" player is deadlocked (it cannot mimic the action). If a strategy exists, it can be represented as a set of traces, each one providing a counterexample to disprove $tg_1 \preceq tg_2$ (cf. Definition 9).

The tool Uppaal TIGA [18] implements the refinement checking for (non-parametric) timed games, which is internally implemented as a strategy synthesis problem. For the parametric case, the tool Romeo [15] allows to perform strategy synthesis for reachability games of timed Petri nets, but it does not primitively support refinement checking, which we conjecture could be encoded manually into a strategy synthesis problem as described above (cf. Section VI).

B. Parametric Timed Games

We now define an extension of timed games called parametric timed games. We start by defining parametric clock constraints. These enhance Definition 1 with the possibility of expressing parameters instead of constants.

Definition 5 (Parametric clock constraint): Let C be a set of clocks and let P be a set of parameters on a clock value domain Δ. Then the set $\Phi_{C,P}$ of *parametric clock constraints* φ on C and P is inductively defined as:

$$\varphi ::= \top \mid c \sim p \mid c \sim n \mid \varphi \wedge \varphi \mid \neg\varphi$$

where $c \in C$, $\sim \in \{<, \leq, \geq, >\}$, and $n \in \Delta$.

We can now define parametric timed games as timed games with parametric clock constraints on both guards and invariants.

Definition 6 (Parametric timed game): A *parametric timed game* is a septuple $(L, \ell_0, \Sigma, C, P, I, E)$, where

- L is a set of *locations*,
- $\ell_0 \in L$ is the *initial location*,
- Σ is a set of *actions*, partitioned into *controllable* actions Σ^c and *uncontrollable* actions Σ^u,
- C is a set of *clocks*,
- P is a set of *parameters*,
- $I : L \to \Phi_{C,P}$ assigns *parametric location invariants*, and
- $E \subseteq L \times \Phi_{C,P} \times \Sigma \times 2^C \times L$ is a set of *transitions*.

Let *PTG* denoted the set of parametric timed games. Finally, let $\pi : \Delta^P \times PTG \mapsto TG$ be the projection function that maps a parameter evaluation σ_P such that $\forall \phi \in \Phi_{C,P} \,.\, \sigma_P \in [\![\phi]\!]$ and a parametric timed game $ptg \in PTG$ into the timed game $\pi(\sigma_P, A) \in TG$ that is obtained by instantiating the parameters in ptg with σ_P.

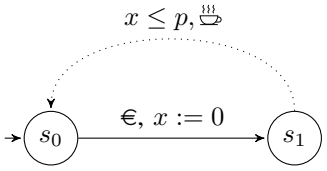

Fig. 2. A parametric timed game *ptg*

Example 2: Continuing Example 1, Figure 2 displays a parametric timed game *ptg* such that $tg = \pi(\sigma_P, ptg)$, with $\sigma_P = \{(p, 4)\}$.

The notion of refinement is extended to the parametric setting by universally quantifying on all parameter evaluations. Refinement is generally undecidable for parametric timed games [17], [24], whereas it is decidable for a fragment of parametric timed games [13]. In the next section, we introduce an extension of this fragment.

IV. Monotonicity of L/U Parametric Timed Games

In this section, we present the first contribution of this paper: we introduce an (extended) fragment of parametric timed games, called L/U parametric timed games, and we prove that a property called *monotonicity* holds.

We start by introducing a novel extended notion of L/U parametric timed game. In L/U parametric timed games, parametric constraints can only appear as upper bound or lower bound (thus the name L/U). In the literature, L/U parametric timed game were defined in [17, Definition 3.3] without permitting parameters in clock constraints of invariants,

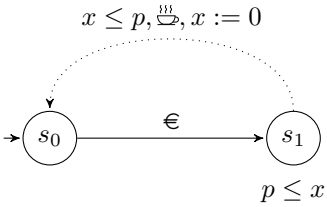

Fig. 3. An (extended) L/U parametric timed game *luptg*

which is what we do allow in the extension we introduce below, while still preserving decidability of refinement. We underline the importance of having a decidable fragment capable of expressing parameters also on invariants, because these are an essential ingredient to impose a form of control over time [14].

Given a parametric timed game *ptg*, parameter p is an upper (or lower, respectively) bound in *ptg* if for each conjunct of each parametric clock constraint in the guards and invariants of *ptg*, $c \sim p$ is either of the form $\sim \in \{<, \leq\}$ (or $\sim \in \{>, \geq\}$, respectively).

Definition 7 ((Extended) L/U parametric timed games): A parametric timed game *ptg* is an extended L/U parametric timed game if every parameter is either an upper bound or a lower bound in *ptg* and the following holds:

- P is partitioned into P^- and P^+.
- Each parameter $p \in P^-$ occurs only as lower bound (upper bound, respectively) in the guards of controllable (uncontrollable, respectively) transitions, and as upper bound (lower bound, respectively) in *Inv*(ℓ) whenever all incoming transitions of ℓ are uncontrollable (controllable, respectively), with $\ell \in L$;
- Each parameter $p \in P^+$ occurs only as upper bound (lower bound, respectively) in the guards of controllable (uncontrollable, respectively) transitions, and as lower bound in *Inv*(ℓ) whenever all incoming transitions of ℓ are uncontrollable, with $\ell \in L$.

From now on we refer to extended L/U parametric timed games as L/U parametric timed games.

Example 3: According to Definition 7, the parametric timed game *ptg* depicted in Figure 2 is an L/U parametric timed game because the only constraint is an upper bound on the uncontrollable transition (thus $p \in P^-$).

The parametric timed game in Figure 3 is an (extended) L/U parametric timed game. Indeed, the parameter $p \in P^-$ appears as a lower bound in the invariant *Inv*$(s_1) = p \leq x$, and the location s_1 has only incoming controllable transitions. Note that the invariant imposes control over time: tea ☕ can only be output by the machine precisely at the instant of time p.

In Section III, we discussed the refinement of parametric timed games, which is undecidable in the general case. Below we show that refinement is decidable for specific L/U parametric timed game models. Indeed, decidability is strictly related to the property of *monotonicity* [13], [17]. We are now ready to state the main result of this section: the property of monotonicity holds for the extended fragment of L/U parametric timed games.

Theorem 1 (Monotonicity of L/U parametric timed games): Let *ptg* $= (L, \ell_0, \Sigma, C, P, I, E)$ be an L/U parametric timed game, let σ_P, σ'_P be such that $\forall p_+ \in P^+, p_- \in P^-.\sigma'_P(p_+) \geq \sigma_P(p_+)$ and $\sigma'_P(p_-) \leq \sigma_P(p_-)$. Then the following holds:

$$\pi(\sigma'_P, ptg) \preceq \pi(\sigma_P, ptg)$$

Proof: Let *tg* $= \pi(\sigma_P, ptg)$ and *tg*$' = \pi(\sigma'_P, ptg)$ be two timed games. We will prove that an enabled controllable transition of *tg* is never disabled in *tg*$'$ and that a disabled uncontrollable transition of *tg* is never enabled in *tg*$'$. We proceed by cases on lower-bound/upper-bound parameters and on controllable/uncontrollable actions. We start from lower bounds $p_- \in P^-$. For each transition t and invariant ℓ we have the following:

- t has action $\alpha \in \Sigma^c$. By Definition 7, its parametric clock constraint (if any) is of the form $p_- \leq c$, where $c \in C_{tg}$. From $\sigma'_P(p_-) \leq \sigma_P(p_-)$ it follows that $\sigma_P(p_-) \leq c \rightarrow \sigma'_P(p_-) \leq c$, hence t is never disabled in *tg*$'$ if it is enabled in *tg*.
- t has action $\alpha \in \Sigma^u$. By Definition 7, its parametric clock constraint (if any) is of the form $p_- \geq c$, where $c \in C_{tg}$. From $\sigma'_P(p_-) \leq \sigma_P(p_-)$ it follows that $\sigma_P(p_-) \not\geq c = \sigma_P(p_-) < c \rightarrow \sigma'_P(p_-) < c = \sigma'_P(p_-) \not\geq c$, hence t is never enabled in *tg*$'$ if it is disabled in *tg*.
- $\ell \in L$ has an invariant $p_- \geq c$, where $c \in C_{tg}$ and all incoming transitions are uncontrollable. From $\sigma'_P(p_-) \leq \sigma_P(p_-)$ it follows that $\sigma_P(p_-) \not\geq c = \sigma_P(p_-) < c \rightarrow \sigma'_P(p_-) < c = \sigma'_P(p_-) \not\geq c$, hence it is never the case that delaying is enabled in *tg*$'$ if it is disabled in *tg*. All states (ℓ, v) that are reachable in *tg* but not in *tg*$'$ by delaying time are out of the refinement relation R by Definition 4. Concerning incident transitions of ℓ, it is never the case that a controllable transition is enabled in *tg* if it is disabled in *tg*$'$ and that an uncontrollable transition is enabled in *tg*$'$ if it is disabled in *tg*.
- $\ell \in L$ has an invariant $p_- \leq c$, where $c \in C_{tg}$ and all incoming transitions are controllable. From $\sigma'_P(p_-) \leq \sigma_P(p_-)$ it follows that $\sigma_P(p_-) \leq c \rightarrow \sigma'_P(p_-) \leq c$, hence it is never the case that delaying is enabled in *tg*$'$ if it is disabled in *tg* (when in ℓ). Concerning incident transitions of ℓ, it is never the case that a controllable transition is enabled in *tg* if it is disabled in *tg*$'$ and that an uncontrollable transition is enabled in *tg*$'$ if it is disabled in *tg*.

Next we consider upper bounds $p_+ \in P$. For each transition t, we now have the following:

- t has action $\alpha \in \Sigma^c$. By Definition 7, its parametric clock constraint (if any) is of the form $p_+ \geq c$, where $c \in C_{tg}$. From $\sigma'_P(p_+) \geq \sigma_P(p_+)$ it follows that $\sigma_P(p_+) \geq c \rightarrow \sigma'_P(p_+) \geq c$, hence t is never disabled in *tg*$'$ if it is enabled in *tg*.

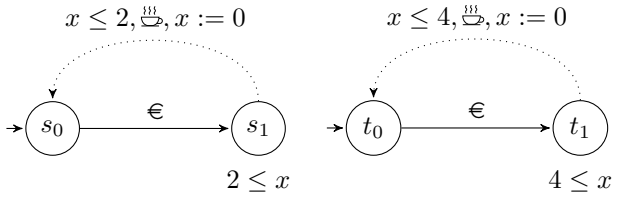

Fig. 4. Two timed games $tg_1 = \pi(\sigma'_P, \mathit{luptg})$ [left] and $tg_2 = \pi(\sigma_P, \mathit{luptg})$ [right], where $\sigma_P = \{(p, 4)\}$, $\sigma'_P = \{(p, 2)\}$, and luptg is given in Figure 3

- t has action $\alpha \in \Sigma^u$. By Definition 7, its parametric clock constraint (if any) is of the form $p_+ \leq c$, where $c \in C_{tg}$. From $\sigma'_P(p_+) \geq \sigma_P(p_+)$ it follows that $\sigma_P(p_+) \not\leq c = \sigma_P(p_+) > c \rightarrow \sigma'_P(p_+) > c = \sigma'_P(p_+) \not\leq c$, hence t is never enabled in *tg′* if it is disabled in *tg*.
- $\ell \in L$ has an invariant $p_+ \leq c$, where $c \in C_{tg}$. From $\sigma'_P(p_+) \geq \sigma_P(p_+)$ it follows that $\sigma_P(p_+) \not\leq c = \sigma_P(p_+) > c \rightarrow \sigma'_P(p_+) > c = \sigma'_P(p_+) \not\leq c$, hence it is never the case that delaying is enabled in *tg′* if it is disabled in *tg*. All states (ℓ, v) that are reachable in *tg* but not in *tg′* by delaying time are out of the refinement relation R by Definition 4. Moreover, by Definition 7, all incoming transitions are uncontrollable. Thus, it is never the case that a controllable transition is enabled in *tg* if it is disabled in *tg′* and that an uncontrollable transition is enabled in *tg′* if it is disabled in *tg*. ■

Example 4: Continuing Examples 1 and 2, and recalling *tg* and *ptg* in Figures 1 and 2, now consider a different parameter evaluation $\sigma'_P = \{(p, 2)\}$. According to Theorem 1, since $\sigma'_P(p) \leq \sigma_P(p)$, it holds that $tg' \preceq tg$, where $tg' = \pi(\sigma'_P, ptg)$.

We now illustrate how the refinement game can be played on L/U parametric timed games.

Example 5: Consider the L/U parametric timed game luptg depicted in Figure 3. Let $\sigma_P = \{(p, 4)\}$, $\sigma'_P = \{(p, 2)\}$, $tg_1 = \pi(\sigma'_P, \mathit{luptg})$, and $tg_2 = \pi(\sigma_P, \mathit{luptg})$. Figure 4 depicts both tg_1 and tg_2. By Theorem 1, it holds that $tg_1 \preceq tg_2$, and we call tg_2 the original system and tg_1 the mutated system. Now we are ready to show how the refinement game can be played.

Firstly, we have a delay from the mutated system tg_1, namely, $(s_0, x = 0) \xrightarrow{2} (s_0, x = 2)$. The original system tg_2 can mimic the delay transition, namely, $(t_0, x = 0) \xrightarrow{2} (t_0, x = 2)$. From $(s_0, x = 2)$, the mutated system could fire its controllable transition $(s_0, x = 2) \xrightarrow{€} (s_1, x = 2)$. However, by Definition 4 the mutated system only plays the uncontrollable output transitions and the delay transitions, whereas the original system plays the controllable input transitions. Hence, from state $(s_0, x = 2)$, the only possibility left for the mutated system is to delay.

Note that in timed games, time cannot be blocked. We have $(s_0, x = 2) \xrightarrow{2} (s_0, x = 4)$. Now the original system tg_2 can mimic this delay transition, namely, $(t_0, x = 2) \xrightarrow{2} (t_0, x = 4)$.

Note that these two delay moves can in fact be represented by one single play, namely $(s_0, x = 0) \xrightarrow{4} (s_0, x = 4)$ and $(t_0, x = 0) \xrightarrow{4} (t_0, x = 4)$. Then, from state $(t_0, x = 4)$, there are two possibilities. If the mutated system continues to delay time from location s_0, the original system will always be capable of mimicking the delay. Otherwise, at some time instant, say $x = 4$, the original system plays the controllable transition $(t_0, x = 4) \xrightarrow{€} (t_1, x = 4)$. The mutated system mimics this transition $(s_0, x = 4) \xrightarrow{€} (s_1, x = 4)$.

Note that the invariant $\mathit{Inv}(s_1) = 2 \leq x$ is satisfied by $(s_1, x = 4)$. From state $(t_1, x = 4)$, the original system could play its uncontrollable output $(t_1, x = 4) \xrightarrow{¥} (t_0, x = 0)$. However, as stated above, the original system only plays the controllable inputs. Hence, from states $(s_1, x = 4)$ and $(t_1, x = 4)$, the only possibility left is that the mutated system delays time. The original system is always capable of mimicking these delays, thus showing that tg_1 is a refinement of tg_2.

V. Monotonicity as Equivalent Mutants Detection

In this section, we present the second contribution of this paper: the relations between model-based mutation testing and L/U parametric timed games, and more specifically, the correspondence between monotonicity and subsumed mutants.

We start by defining the abstraction of a timed game into a parametric timed game that is obtained by turning all constants of clock constraints of transitions and invariants into distinct parameters. This is necessary since the system under test is modelled as a timed game, and abstracting to a corresponding parametric timed game is thus necessary to perform symbolic reasoning on parameter evaluations. We denote with $e_{(i)}$ the projection of a tuple e on its ith element, where indices start from zero (invariants are pairs, whereas transitions are quintuples) and $e' = e[u'/u]$ is such that for all j, if $e_{(j)} = u$, then $e'_{(j)} = u'$ else $e'_{(j)} = e_{(j)}$.

Definition 8 (Parametric abstraction): Let $tg = (L, \ell_0, \Sigma, C, I, E)$ be a timed game and let $M = \{\, ti \mid ti \in E \cup I,\ ti_{(1)} = c \sim n \in \Phi_C \,\}$. We denote with $\pi^{-1}(tg) = (L, \ell_0, \Sigma, C, P, I', E')$ the *abstraction* of *tg* into a parametric timed game such that there exists a bijection $f : M \mapsto P$ such that $E' = (E \setminus M) \cup \{\, t[c \sim p/c \sim n] \mid t \in M \cap E,\ p = f(t) \,\}$ and $I' = (I \setminus M) \cup \{\, i[c \sim p/c \sim n] \mid i \in M \cap I,\ p = f(i) \,\}$.

Note that since an element $c \sim n \in \Phi_C$ could be shared by different invariants and transitions of the timed game, the injection guarantees that each parameter is unique to a transition or invariant in the abstracted parametric timed game.

Example 6: Continuing the previous examples, we have that $ptg = \pi^{-1}(tg)$, i.e., the parametric timed game in Figure 2 can be obtained by applying the parametric abstraction on the timed game in Figure 1.

The six mutation operators that were used in [11], [12] to generate mutants of (featured) timed games are displayed in Figure 5. Complex mutation functions not only decrease or increase parameter valuations, but can also remove or add locations and transitions. These mutations can be encoded into a parametric timed game with further transitions, parameters,

Mutation Operators

TMI	Transition MIssing operator removes a transition
TAD	Transition ADd operator adds a transition between two locations
LMI	Location MIssing operator removes a location (other than the initial location) and all its incident transitions
CXL	Constant eXchange L operator increases the constant of a clock constraint
CXS	Constant eXchange S operator decreases the constant of a clock constraint
CCN	Clock Constraint Negation operator negates a clock constraint

Fig. 5. The mutation operators from [11], [12]

and guards to enable to remove or add locations and transitions by only using these parameter valuations. However, adding such guards is not possible for the class of L/U parametric timed games for which Theorem 1 holds. On the other hand, mutations CXL and CXS are suitable to be modelled as parametric constraints ϕ, similarly to how this was done in [12].

Example 7: Continuing the previous examples, timed game tg' is a mutant of timed game tg that is obtained by applying the mutation operator CXS on the parameter evaluation σ_P of its abstracted parametric timed game, denoted as $m_{\text{CXS},p,2}(\sigma_P) = \sigma'_P$, where $m_{\text{CXS},p,2}$ is a mutation function decreasing by 2 units the evaluation of parameter p. In the same way, tg_1 is a mutant of tg_2 depicted in Figure 4.

Note that in [11], [12], the notion of refinement was used to characterise when a mutation yields a subsumed mutant. Indeed, a mutation modifies the behaviour of a system. If such mutation produces a refinement, this means that a tester cannot distinguish the mutated system from its original counterpart (i.e., the mutated system is subsumed). In fact, our notion of refinement (cf. Definition 4) constrains the refined timed game to never increase the uncontrollable behaviour, whereas the controllable behaviour cannot be reduced. This is because for the purpose of testing, the point of view of the environment is considered. The refined version must still make all inputs (i.e., controllable actions) available to the environment, whereas its outputs can be reduced.

For example, a beverage machine only accepting euro coins is not a refinement of a machine accepting either euro or dollar coins. Indeed, a test for the original machine would fail if dollar coins would be inserted in the machine that only accepts euro coins. In mutation testing jargon, the machine only accepting euro coins is a mutant not subsumed by the original system and the test not passing on the mutated system is said to be "killing" that mutant. On the other hand, a machine that (non-deterministically) outputs a tea or a coffee when a euro coin is inserted can be refined into a machine that always outputs a coffee when a euro coin is inserted. Since a test for the less refined machine must handle both tea and coffee, the test will not fail if coffee is returned. Moreover, in this case the external tester of the machine cannot distinguish whether it is interacting with the original machine, which in this case is always deciding to output a coffee but could at some point outputs a tea, or a refined machine that always returns a coffee. Concerning timing behaviour, a refined timed game is not allowed to perform slower than its unrefined version.

Below we formally define a counterexample to prove that a refinement between two timed games does not hold, this counterexample can be used to distinguish the refined version from the original. When the mutated version is not subsumed, the generated trace will be capable of detecting the mutant as different from the original system (i.e., killing the mutant). The generated counterexample trace is composed of a series of inputs to the system, outputs from the system, and delays that can be turned into a test, following model-based testing principles [1], [32], [33]. The test will pass on the original system and will fail on the mutant, as expected.

Definition 9 (Counterexample): Let a counterexample that shows that $tg_1 \preceq tg_2$ does not hold be a timed trace $w\alpha$ such that $w \in [\![tg_1]\!] \cap [\![tg_2]\!]$, and either (i) $\alpha \in \Sigma_2^u$ and $w\alpha \in [\![tg_2]\!] \setminus [\![tg_1]\!]$, or (ii) $\alpha \in \Sigma_1^c \cup \Delta$ and $w\alpha \in [\![tg_1]\!] \setminus [\![tg_2]\!]$. Now let $Ctx(tg_1, tg_2) \subseteq (\Sigma_1 \cup \Sigma_2 \cup \Delta)^*$ be the set of counterexamples that show that $tg_1 \not\preceq tg_2$, where $Ctx(tg_1, tg_2) = \emptyset$ if and only if $tg_1 \preceq tg_2$.

A. Mutation Testing with Parametric Timed Games

We now discuss the steps that are required to perform mutation testing with parametric timed games. The starting point is a timed game *tg* modelling the system (several case studies modelled as timed games have been analysed in [12]). To allow symbolic reasoning on mutations, the timed game must be abstracted into a parametric timed game *ptg* using Definition 8. This operation also yields the parameter evaluation σ_P such that by projecting the parametric timed game on σ_P the original timed game is obtained. The mutation function can thus be applied to mutate the parameter evaluation σ_P, obtaining a mutated parameter evaluation $\sigma_{P'}$. At this point, using refinement as defined in Definition 4, it is possible to detect whether the projection of parametric timed game *ptg* on $\sigma_{P'}$ produces a mutant subsumed by the projection of *ptg* on σ_P (i.e., the timed game *tg*).

Finally, after collecting a set of mutations that are not refinements of the original system, a suite of tests capable of "killing" all the generated mutants is computed automatically (cf. Definition 9). Indeed, in mutation testing the strength of a test-suite is measured as the ratio between the mutants killed by the test-suite and the total amount of mutants generated.

Thus, by construction, the algorithm produces a test-suite with top strength, since it is capable of killing all generated mutants.

We summarise below how to perform mutation testing using parametric timed games. Note that the test for mutant equivalence is emphasised in bold, and it is targeted below. Indeed, to test whether $m \notin$ *EquMut* (i.e., the mutant is not subsumed), we can exploit the monotonicity result in the following way: if the conditions of Theorem 1 are met, then we know by construction that the mutant is a refinement and can be discarded.

Generating a test-suite for a timed game

1) Let *tg* be a timed game such that $ptg = \pi^{-1}(tg)$ is the abstracted parametric timed game, and σ_P is the parameter evaluation such that $tg = \pi(\sigma_P, ptg)$. Repeat the following steps.
2) Let $m : \Delta^P \mapsto \Delta^P$ be a *mutation* and let $\sigma_{P'} = m(\sigma_P)$ be the *mutant*.
3) Let $\pi(\sigma'_P, tg)$ be *subsumed* by $\pi(\sigma_P, ptg)$ if and only if $\pi(\sigma'_P, ptg) \preceq \pi(\sigma_P, ptg)$, and let *EquMut* be the set of mutation functions yielding subsumed mutants. **Select $m \notin$ *EquMut***.
4) Add $Ctx(\pi(\sigma'_P, ptg), \pi(\sigma_P, ptg))$ to the test-suite under construction.

Example 8: Continuing the previous examples, note that it holds that $m_{\texttt{CXS},p,2} \in EquMut$. Next, consider a mutation $m_{\texttt{CXL},p,2}(\sigma_P) = \sigma''_P$, where $\sigma''_P = \{(p, 6)\}$. In this case, we obtain that $tg'' \not\preceq tg$ for the timed game $tg'' = \pi(\sigma''_P, ptg)$. A counterexample $w\alpha$ (cf. Definition 9) is such that $w = €5$ and $\alpha = \overset{\text{!!!}}{\smile}$. Indeed, it holds that $w\alpha \in [\![tg'']\!] \setminus [\![tg]\!]$ and $w\alpha \in Ctx(tg'', tg)$. The trace $w\alpha$ can be turned into a test by construction capable of detecting the mutant.

B. Correspondence between Monotonicity and Commandments

We now prove the correspondence between monotonicity and violation of a subset of seven commandments on model-based mutation testing for timed games, which are reported in Figure 6, and which have been selected from those that have been presented in [11], [12]. A violation of such a commandment implies that the corresponding mutant is subsumed, as is proved in [12, Lemma 4-7]. It is worth mentioning that by using the framework of L/U parametric timed games, we simplify considerably the theory that has been presented in [11], [12]. Notably, the aforementioned four statements reported in [12, Lemmata 4-7] are all instances of Theorem 2 below. By restricting to L/U parametric timed games, it is possible to statically detect conditions for m that imply $m \in$ *EquMut*, by using Theorem 1, thus avoiding semantics checking of the refinement relation, as proved next.

Theorem 2 (Monotonicity as Equivalent Mutants Detection): Let *tg* be a timed game and let *ptg* be an L/U parametric timed game obtained from *tg* using Definition 8 such that $tg = \pi(\sigma_P, ptg)$. Let m be a mutation of *ptg*, where m is either CXL or CXS, and let $\sigma'_P = m(\sigma_P)$ be the *mutant*. Consider the 7 commandments in Figure 6. Then the following holds:

m violates one of the 7 commandments if and only if
$\forall p_+ \in P^+,\ p_- \in P^-.\sigma'_P(p_+) \geq \sigma_P(p_+)$ and $\sigma'_P(p_-) \leq \sigma_P(p_-)$

Proof: The proof proceeds by cases on the two mutation operators CXL and CXS and the corresponding commandments reported in Figure 6:

- CXL: recall from Figure 5 that this mutation operator increases the constant of a clock constraint of the timed game. The corresponding commandments are as follows:
  - *CXL shall not be applied to controllable transitions with guards of the form* $x \leq k$: by Definition 7, parameters $p \in P^+$ occur in controllable transitions with guards of the form $x \leq p$. It follows that this commandment is violated if and only if $\sigma'_P(p) \geq \sigma_P(p)$.
  - *CXL shall not be applied to uncontrollable transitions with guards of the form* $x \geq k$: by Definition 7, parameters $p \in P^+$ occur in uncontrollable transitions with guards of the form $x \geq p$. It follows that this commandment is violated if and only if $\sigma'_P(p) \geq \sigma_P(p)$.
  - *CXL shall not be applied to invariants of the form* $x \geq k$ *whenever all incoming transitions are uncontrollable*: by Definition 7, parameters $p \in P^+$ occur in invariants of the form $x \geq p$ whenever all incoming transitions are uncontrollable. It follows that this commandment is violated if and only if $\sigma'_P(p) \geq \sigma_P(p)$.
- CXS: recall from Figure 5 that this mutation operator decreases the constant of a clock constraint of the timed game. The corresponding commandments are as follows:
  - *CXS shall not be applied to controllable transitions with guards of the form* $x \geq k$: by Definition 7, parameters $p \in P^-$ occur in controllable transitions with guards of the form $x \geq p$. It follows that this commandment is violated if and only if $\sigma'_P(p) \leq \sigma_P(p)$.
  - *CXS shall not be applied to uncontrollable transitions with guards of the form* $x \leq k$: by Definition 7, parameters $p \in P^-$ occur in uncontrollable transitions with guards of the form $x \leq p$. It follows that this commandment is violated if and only if $\sigma'_P(p) \leq \sigma_P(p)$.
  - *CXS shall not be applied to invariants of the form* $x \leq k$ *whenever all incoming transitions are uncontrollable*: by Definition 7, parameters $p \in P^-$ occur in invariants of the form $x \leq p$ whenever all incoming transitions are uncontrollable. It follows that this commandment is violated if and only if $\sigma'_P(p) \leq \sigma_P(p)$.

Commandments of Model-Based Mutation Testing:

CXL shall not be applied to controllable transitions with guards of the form $x \leq k$

CXL shall not be applied to uncontrollable transitions with guards of the form $x \geq k$

CXL shall not be applied to invariants of the form $x \geq k$ *whenever all incoming transitions are uncontrollable*

CXS shall not be applied to controllable transitions with guards of the form $x \geq k$

CXS shall not be applied to uncontrollable transitions with guards of the form $x \leq k$

CXS shall not be applied to invariants of the form $x \leq k$ *whenever all incoming transitions are uncontrollable*

CXS shall not be applied to invariants of the form $x \geq k$ *whenever all incoming transitions are controllable*

where x is a clock constraint and k is a constant

Fig. 6. The subset of 7 out of 10 commandments from [12] used in Theorem 2

– *CXS shall not be applied to invariants of the form* $x \geq k$ *whenever all incoming transitions are controllable*: by Definition 7, parameters $p \in P^-$ occur in invariants of the form $x \geq p$ whenever all incoming transitions are controllable. It follows that this commandment is violated if and only if $\sigma'_P(p) \leq \sigma_P(p)$. ■

Note that the converse of Theorem 1 does not hold: if a mutation does not meet the hypotheses of Theorem 1 (or, according to Theorem 2, it does not violate one of the above commandments), then it is not always true that the mutant is not a refinement. For example, it suffices to apply one such mutation on a transition that is never enabled. Thus, Theorem 1 does not completely characterise the set *EquMut*.

Indeed, whilst it is syntactically possible to detect mutants that are refinements, to detect mutants that are not refinements, one in general needs additional information that is not easily retrievable syntactically [12]. Notwithstanding the above obstacle, according to the literature, the number of syntactically detected equivalent mutants (using the 10 commandments presented in [12]), can be up to $\approx 80\%$ of all equivalent mutants generated by randomly applying mutations, as experimentally validated in [12] (cf. also the experiments reproducibility package [34]). Of course, if only mutations satisfying the hypothesis of Theorem 1 are applied, then 100% of all mutants are detected as subsumed.

Example 9: Continuing the previous examples, consider the mutant $\sigma'_P = \{(p, 2)\}$, which mutates $\sigma_P = \{(p, 4)\}$. As previously stated, following Theorem 1, since $\sigma'_P(p) \leq \sigma_P(p)$, it holds that $tg' \preceq tg$, where $tg' = \pi(\sigma'_P, ptg)$.

This is also confirmed by applying Theorem 2, because this mutation violates a commandment from [12]. In particular, p occurs as guard $x \leq p$ of an uncontrollable (output) of the tea machine, and p is decreased in the mutant. Hence, the applied mutation operator is CXS (decreasing the constant of a clock constraint). This mutation indeed violates one of the commandment reported in [12], viz. the $5th$ commandment from Figure 6:

CXS shall not be applied to uncontrollable transitions with guards of the form $x \leq k$

By Theorem 2, this is possible if and only if $\sigma'_P(p) \leq \sigma_P(p)$, where $p \in P^-$.

Remark 1: Note that, thanks to Theorem 2, the validation of the commandments as performed in [12] (and described above) can be imported as a validation of the approach proposed in this paper, namely, the detection of equivalent mutants through the monotonicity property of the extended L/U parametric timed games fragment. Moreover, to further emphasise the importance of expressing invariants in timed games, we note that all the real-world benchmark case studies addressed in [12] contain invariants in their models.

VI. Conclusion

We formally proved a correspondence between violations of some commandments for detecting equivalent mutants presented in the literature [11], [12] and the monotonicity property for a fragment of parametric timed games. To this aim, we extended the L/U parametric times gamed fragment to be able to express parametric constraints on invariants, whilst maintaining the monotonicity property that is essential for the decidability of key problems for this fragment [17]. The introduced invariants on parametric constraints are an essential ingredient to impose a form of control on time [14]. These results pave the way to the use of theories and tools created in the framework of parametric timed games for model-based mutation testing, allowing to detect equivalent mutants and at the same time providing new research goals, some of which we discuss next.

First, while tools for parametric timed games are available (e.g., [15], [16]), none of them performs refinement checking, which is however useful for model-based mutation testing, as we showed in this paper. As discussed in the related work, this could be obtained with a suitable encoding of the refinement checking problem for parametric timed games into the problem of strategy synthesis for reachability games, as was done in [14] for the non-parametric case. With such an encoding, it could be possible to reuse the algorithms and the tool Romeo as recently

presented in [15], [19], to perform refinement checking for parametric timed games. However, since Romeo supports both timed Petri nets and clock transition systems [35], a suitable conversion to these formalisms becomes necessary [35], [36].

Another open research goal concerns extending parametric timed games to include configurations. The mapping from configurable parametric timed automata to parametric timed automata (cf. [37, Def 3.8–3.9]) is trivially extensible to games. Indeed, the encoding relies on a mapping from each feature to a Boolean parameter denoting its activation/deactivation, and by adding a new *urgent* initial location (i.e., in which delays are not allowed) with only one outgoing controllable transition to the original initial location. This added transition will be guarded by the parametric constraint encoding the feature model, and by adding the featured clock constraints in conjunction with the other constraints of the other transitions. Configurations for parametric timed games would allow to express in a compact way all the applicable mutations, in the style of featured mutant models [8]. This could help to extend the presented theory to other types of mutations (e.g., removing transitions or locations), as well as applying strategy synthesis *all-at-once* for all possible selected mutations, as recently studied in [38] for the non-parametric case.

ACKNOWLEDGMENT

The first two authors acknowledge funding from the MUR PRIN 2020TL3X8X project T-LADIES (Typeful Language Adaptation for Dynamic, Interacting and Evolving Systems).

REFERENCES

[1] M. Utting, A. Pretschner, and B. Legeard, "A Taxonomy of Model-Based Testing Approaches," *Softw. Test. Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, 2012. [Online]. Available: https://doi.org/10.1002/stvr.456

[2] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 608–624, 2006. [Online]. Available: https://doi.org/10.1109/TSE.2006.83

[3] J. Offutt, "A mutation carol: Past, present and future," *Inf. Softw. Technol.*, vol. 53, no. 10, pp. 1098–1107, 2011. [Online]. Available: https://doi.org/10.1016/j.infsof.2011.03.007

[4] B. K. Aichernig, H. Brandl, E. Jöbstl, W. Krenn, R. Schlick, and S. Tiran, "Killing Strategies for Model-Based Mutation Testing," *Softw. Test. Verif. Reliab.*, vol. 25, no. 8, pp. 716–748, 2015. [Online]. Available: https://doi.org/10.1002/stvr.1522

[5] A. Brillout, N. He, M. Mazzucchi, D. Kroening, M. Purandare, P. Rümmer, and G. Weissenbacher, "Mutation-Based Test Case Generation for Simulink Models," in *Proceedings of the 8th International Symposium on Formal Methods for Components and Objects (FMCO'09)*, ser. LNCS, F. S. de Boer, M. M. Bonsangue, S. Hallerstede, and M. Leuschel, Eds., vol. 6286. Springer, 2009, pp. 208–227. [Online]. Available: https://doi.org/10.1007/978-3-642-17071-3_11

[6] R. Baker and I. Habli, "An Empirical Evaluation of Mutation Testing for Improving the Test Quality of Safety-Critical Software," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 787–805, 2013. [Online]. Available: https://doi.org/10.1109/TSE.2012.56

[7] B. K. Aichernig, F. Lorber, and D. Nickovic, "Time for mutants: Model-based mutation testing with timed automata," in *Proceedings of the 7th International Conference on Tests and Proofs (TAP'13)*, ser. LNCS, M. Veanes and L. Viganò, Eds., vol. 7942. Springer, 2013, pp. 20–38. [Online]. Available: https://doi.org/10.1007/978-3-642-38916-0_2

[8] X. Devroey, G. Perrouin, M. Papadakis, A. Legay, P. Schobbens, and P. Heymans, "Featured Model-based Mutation Analysis," in *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*. ACM, 2016, pp. 655–666. [Online]. Available: https://doi.org/10.1145/2884781.2884821

[9] K. G. Larsen, F. Lorber, B. Nielsen, and U. M. Nyman, "Mutation-Based Test-Case Generation with Ecdar," in *Proceedings of the 10th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW'17)*. IEEE, 2017, pp. 319–328. [Online]. Available: https://doi.org/10.1109/ICSTW.2017.60

[10] M. Kintis, M. Papadakis, Y. Jia, N. Malevris, Y. L. Traon, and M. Harman, "Detecting trivial mutant equivalences via compiler optimisations," *IEEE Trans. Softw. Eng.*, vol. 44, no. 4, pp. 308–333, 2018. [Online]. Available: https://doi.org/10.1109/TSE.2017.2684805

[11] D. Basile, M. H. ter Beek, M. Cordy, and A. Legay, "Tackling the Equivalent Mutant Problem in Real-Time Systems: The 12 Commandments of Model-Based Mutation Testing," in *Proceedings of the 24th ACM Conference on Systems and Software Product Lines (SPLC'20)*. ACM, 2020, pp. 252–262. [Online]. Available: https://doi.org/10.1145/3382025.3414966

[12] D. Basile, M. H. ter Beek, S. Lazreg, M. Cordy, and A. Legay, "Static detection of equivalent mutants in real-time model-based mutation testing: An Empirical Evaluation," *Empir. Softw. Eng.*, vol. 27, no. 7, pp. 160:1–160:55, 2022. [Online]. Available: https://doi.org/10.1007/s10664-022-10149-y

[13] A. Jovanovic, S. Faucou, D. Lime, and O. H. Roux, "Real-time control with parametric timed reachability games," *IFAC Proc. Vol.*, vol. 45, no. 29, pp. 323–330, 2012, Proceedings of the 11th IFAC Workshop on Discrete Event Systems (WODES'12). [Online]. Available: https://doi.org/10.3182/20121003-3-MX-4033.00052

[14] A. David, K. G. Larsen, A. Legay, U. Nyman, L. Traonouez, and A. Wasowski, "Real-time specifications," *Int. J. Softw. Tools Technol. Transf.*, vol. 17, no. 1, pp. 17–45, 2015. [Online]. Available: https://doi.org/10.1007/s10009-013-0286-x

[15] D. Lime, O. H. Roux, C. Seidner, and L.-M. Traonouez, "Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches," in *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09)*, ser. LNCS, S. Kowalewski and A. Philippou, Eds., vol. 5505. Springer, 2009, pp. 54–57. [Online]. Available: https://doi.org/10.1007/978-3-642-00768-2_6

[16] É. André, "IMITATOR 3: Synthesis of Timing Parameters Beyond Decidability," in *Proceedings of the 33rd International Conference on Computer Aided Verification (CAV'21)*, ser. LNCS, A. Silva and K. R. M. Leino, Eds., vol. 12759. Springer, 2021, pp. 552–565. [Online]. Available: https://doi.org/10.1007/978-3-030-81685-8_26

[17] A. Jovanović, D. Lime, and O. H. Roux, "A game approach to the parametric control of real-time systems," *Int. J. Control*, vol. 92, no. 9, pp. 2025–2036, 2019. [Online]. Available: https://doi.org/10.1080/00207179.2018.1426883

[18] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "UPPAAL-Tiga: Time for Playing Games!" in *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07)*, ser. LNCS, W. Damm and H. Hermanns, Eds., vol. 4590. Springer, 2007, pp. 121–125. [Online]. Available: https://doi.org/10.1007/978-3-540-73368-3_14

[19] A. Jovanovic, D. Lime, and O. H. Roux, "Control of real-time systems with integer parameters," *IEEE Trans. Autom. Control.*, vol. 67, no. 1, pp. 75–88, 2022. [Online]. Available: https://doi.org/10.1109/TAC.2020.3046578

[20] W. Zuberek, "Timed Petri nets definitions, properties, and applications," *Microelectron. Reliab.*, vol. 31, no. 4, pp. 627–644, 1991. [Online]. Available: https://doi.org/10.1016/0026-2714(91)90007-T

[21] L. Luthmann, T. Gerecht, and M. Lochau, "Sampling strategies for product lines with unbounded parametric real-time constraints," *Int. J. Softw. Tools Technol. Transf.*, vol. 21, no. 6, pp. 613–633, 2019. [Online]. Available: https://doi.org/10.1007/s10009-019-00532-4

[22] L. Luthmann, A. Stephan, J. Bürdek, and M. Lochau, "Modeling and Testing Product Lines with Unbounded Parametric Real-Time Constraints," in *Proceedings of the 21st International Systems and Software Product Lines Conference (SPLC'17)*. ACM, 2017, pp. 104–113. [Online]. Available: https://doi.org/10.1145/3106195.3106204

[23] É. André, P. Arcaini, A. Gargantini, and M. Radavelli, "Repairing Timed Automata Clock Guards through Abstraction and Testing," in *Proceedings of the 13th International Conference on Tests and Proofs (TAP@FM'19)*, ser. LNCS, D. Beyer and C. Keller, Eds., vol. 11823. Springer, 2019, pp. 129–146. [Online]. Available: https://doi.org/10.1007/978-3-030-31157-5_9

[24] É. André, "What's decidable about parametric timed automata?" *Int.*

J. Softw. Tools Technol. Transf., vol. 21, no. 2, pp. 203–219, 2019. [Online]. Available: https://doi.org/10.1007/s10009-017-0467-0

[25] B. Khaireddine, A. Zakharchenko, and A. Mili, "A Generic Algorithm for Program Repair," in *Proceedings of the 5th IEEE/ACM International FME Workshop on Formal Methods in Software Engineering (FormaliSE'17)*. IEEE, 2017, pp. 65–71. [Online]. Available: https://doi.org/10.1109/FormaliSE.2017.7

[26] S. Hallé, "Test Suite Generation for Boolean Conditions with Equivalence Class Partitioning," in *Proceedings of the 10th IEEE/ACM International Conference on Formal Methods in Software Engineering (FormaliSE'22)*. ACM, 2022, pp. 23–33. [Online]. Available: https://doi.org/10.1145/3524482.3527659

[27] A. Knüppel, L. Schaer, and I. Schaefer, "How much Specification is Enough? Mutation Analysis for Software Contracts," in *Proceedings of the 9th IEEE/ACM International Conference on Formal Methods in Software Engineering (FormaliSE'21)*. IEEE, 2021, pp. 42–53. [Online]. Available: https://doi.org/10.1109/FormaliSE52586.2021.00011

[28] R. Alur and D. L. Dill, "A Theory of Timed Automata," *Theoret. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994. [Online]. Available: https://doi.org/10.1016/0304-3975(94)90010-8

[29] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, "Controller Synthesis for Timed Automata," *IFAC Proc. Vol.*, vol. 31, no. 18, pp. 447–452, 1998, Proceedings of the 5th IFAC Conference on System Structure and Control (SSC'98). [Online]. Available: https://doi.org/https://doi.org/10.1016/S1474-6670(17)42032-5

[30] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Timed I/O Automata: A Complete Specification Theory for Real-time Systems," in *Proceedings of the 13th International Conference on Hybrid Systems: Computation and Control (HSCC'10)*. ACM, 2010, pp. 91–100. [Online]. Available: https://doi.org/10.1145/1755952.1755967

[31] M. Krichen and S. Tripakis, "Conformance testing for real-time systems," *Form. Methods Syst. Des.*, vol. 34, no. 3, pp. 238–304, 2009. [Online]. Available: https://doi.org/10.1007/s10703-009-0065-1

[32] L. B. Briones and E. Brinksma, "Testing Real-Time Multi Input-Output Systems," in *Proceedings of the 7th International Conference on Formal Engineering Methods (ICFEM'05)*, ser. LNCS, K. Lau and R. Banach, Eds., vol. 3785. Springer, 2005, pp. 264–279. [Online]. Available: https://doi.org/10.1007/11576280_19

[33] M. Bouwman, D. van der Wal, B. Luttik, M. Stoelinga, and A. Rensink, "A Case in Point: Verification and Testing of a EULYNX Interface," *Form. Asp. Comput.*, 2023. [Online]. Available: https://doi.org/10.1145/3528207

[34] D. Basile, "Experiments Reproducibility package for the paper Static Detection of Equivalent Mutants in Real-Time Model-based Mutation Testing: An Empirical Evaluation," December 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5749732

[35] C. Jard, D. Lime, and O. H. Roux, "Blending timed formal models with clock transition systems," *Fundam. Inform.*, vol. 129, no. 1-2, pp. 85–100, 2014. [Online]. Available: https://doi.org/10.3233/FI-2014-962

[36] F. Cassez and O. H. Roux, "Structural translation from time Petri nets to timed automata," *J. Syst. Softw.*, vol. 79, no. 10, pp. 1456–1468, 2006. [Online]. Available: https://doi.org/10.1016/j.jss.2005.12.021

[37] L. Luthmann, "Specification and Analysis of Software Systems with Configurable Real-Time Behavior," Ph.D. dissertation, Technische Universität Darmstadt, 2020. [Online]. Available: http://tuprints.ulb.tu-darmstadt.de/17363/

[38] U. Fahrenberg and A. Legay, "Featured games," *Sci. Comput. Program.*, vol. 223, 2022. [Online]. Available: https://doi.org/10.1016/j.scico.2022.102874