A Parametric Counterexample Refinement Approach for Robust Timed Specifications

Louis-Marie Traonouez

A Parametric Counterexample Refinement Approach for Robust Timed Specifications

Louis-Marie Traonouez
Aalborg University, Denmark
lmtr@cs.aau.dk

Robustness analyzes the impact of small perturbations in the semantics of a model. This allows to model hardware imprecision and therefore it has been applied to determine implementability of timed automata. In a recent paper, we extend this problem to a specification theory for real-timed systems based on timed input/output automata, that are interpreted as two-player games. We propose a construction that allows to synthesize an implementation of a specification that is robust under a given timed perturbation, and we study the impact of these perturbations when composing different specifications.

To complete this work we present a technique that evaluates the greatest admissible perturbation. It consists in an iterative process that extracts a spoiling strategy when a game is lost, and through a parametric analysis refines the admissible values for the perturbation. We demonstrate this approach with a prototype implementation.

1 Introduction

Component-based design is a software development paradigm well established in the software engineering industry. In component-based design, larger systems are built from smaller modules that depend on each other in well delimited ways described by interfaces. The use of explicit interfaces encourages creation of robust and reusable components. *Specification theories* provide a language for specifying component interfaces together with operators for combining them, such as parallel composition, along with algorithms for verification based on refinement checking.

For real-time systems, timed automata [5] are the classical specification language. Designs specified as timed automata are traditionally validated using model-checking against correctness properties expressed in a suitable timed temporal logic [17]. Mature modeling and model-checking tools exist, such as Uppaal [9], that implement this technique and have been applied to numerous industrial applications.

In [15], the authors proposed a specification theory for real time systems, based on an input/output extension of timed automata model to specify both models and properties. It uses refinement checking instead of model-checking to support compositionality of designs and proofs from ground up. The set of state transitions of the timed systems is partitioned between inputs, representing actions of the environment, and outputs that represent the behaviour of the component. The theory is equipped with a game-based semantic. The two players, Input and Output, compete in order to achieve a winning objective—for instance safety or reachability.

The theory of [15] is equipped with a compatibility check and a consistency check that allows to decide whether a specification can indeed be implemented. Unfortunately, this check does not take limitations and imprecision of the physical world into account. This is best explained with an example. Consider the specification of a coffee machine in Fig. 1. This machine first ask for the choice of a drink, then awaits a coin, and after receiving the payment it delivers the coffee. If the payment does not arrive

To appear in EPTCS.

within 6 time units, the machine aborts the drink selection and returns to the initial state, awaiting a new choice of a beverage. Already in this simple example it is quite hard to see, that implementing a component satisfying this specification is not quite possible due to a subtle mistake. Observe that the two first steps of the machine are controlled by the environment, and not the system itself. Thus any implementation has to be able to accept the following behaviour: first choice? and then the coin? arriving precisely 6 time units after the choice. However then we arrive at the state $(\mathsf{Serving}, y = 6)$ which requires that the coffee (cof!) must be delivered immediately, in zero time. No physical system would permit this, so we say that this state is not robustly consistent.

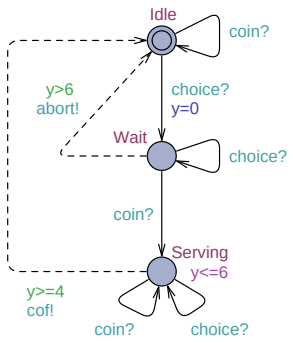


Figure 1: Non robust specification of a coffee machine

The above example can be fixed easily by adding another reset to clock y, when the coin? message is received. It is probably the intended behaviour of the specification that the serving should take 6 time units from the insertion of the coin, and not from the choice of the drink. Finding such errors in specifications is even harder in larger designs as non-robust timing can emerge in the compositions of multiple specifications, as a result of combing behaviours that themselves are robust.

The timing precision errors in specifications are not handled in any way in idealized interface theories such as [15, 4]. These and similar issues have let to a definition of the so called timing *robustness problem* that checks if a model can admit some timing perturbations while preserving a desired property. The robustness problem has been studied in various works for timed automata and it has been linked to the implementability problem [25]. In [20], we extend the specification theory of [15] to support robustness analysis. We check robust consistency and robust compatibility under the assumption of a given small perturbation. However, we were not able to decide if any perturbation can be admitted, neither determine the maximum amount. That is the goal of this paper, to address the parametric problems for robust consistency and robust compatibility. Our contributions include:

- We present a technique that evaluates the greatest admissible perturbation for the robustness problems. We apply a counterexample abstraction refinement-like technique, that analyzes parametrically the results of lost timed games in order to refine the value of the perturbation.
- We introduce a prototype tool that implements this technique and some other functionalities from the theory of [20].
- We demonstrate the performances compared to a simple binary search technique for finding an optimal precision value.

Related works The robust semantics for timed automata with clock drifts has been introduced by Puri [22]. The problem has been linked to the implementation problem in [25], which introduced the first semantics that modeled the hardware on which the automaton is executed. In this work, the authors proposed a robust semantics of Timed Automata called AASAP semantics (for "Almost As Soon As Possible"), that enlarges the guards of an automaton by a delay Δ. This work has been extended in [24] that proposes another robust semantics with both clock drifts and guard enlargement. Extending [22] they solve the robust safety problem, defined as the existence of a non-null value for the imprecision. They show that in terms of robust safety the semantics with clock drifts is just as expressive as the semantics with delay perturbation.

Robust timed games have been studied in [13]. In [20], we adapt their technique to check robust consistency and robust compatibility.

Robustness is defined in [24] as the existence of a positive value for the imprecision of a timed automata. They prove that this problem is decidable, but they do not synthesize the value. A bound on the value is computed in [10]. Finally a quantitative analysis is performed in [19] that computes the greatest admissible value for the perturbation, but the method is restricted to timed automata without nested loops. We propose an approximation technique that evaluates this value in the context of timed specifications, with no major restrictions on syntax of the specifications.

Organization of the paper: We introduce in Section 2 basic definitions for timed systems and timed games. In Section 3 we recall the theory of robust timed specifications describe in [20] and [15]. The main contribution of this paper comes in Section 4, with a counterexample refinement technique to measure the imprecision allowed by the specifications. We present in Section 5 a tool that implements this technique, and we demonstrate its performances in Section 6.

2 Preliminaries

We use $\mathbb{N}$ for the set of all non-negative integers, $\mathbb{R}$ for the set of all real numbers, and $\mathbb{R}_{\geq 0}$ (resp. $\mathbb{R}_{>0}$) for the non-negative (resp. strictly positive) subset of $\mathbb{R}$. Rational numbers are denoted by $\mathbb{Q}$, and their subsets are denoted analogously.

In the framework of [15], specifications and their implementations are semantically represented by Timed I/O Transition Systems (TIOTS) that are nothing more than timed transition systems with input and output modalities on transitions. Input represents the behaviours of the environment in which a specification is used, while output represents behaviours of the component itself.

Definition 1 *A* Timed I/O Transition System *is a tuple $S = (St^S, s_0, \Sigma^S, \rightarrow^S)$, where St^S is an infinite set of states, $s_0 \in St^S$ is the initial state, $\Sigma^S = \Sigma_i^S \oplus \Sigma_o^S$ is a finite set of actions partitioned into inputs Σ_i^S and outputs Σ_o^S, and $\rightarrow^S$: $St^S \times (\Sigma^S \cup \mathbb{R}_{\geq 0}) \times St^S$ is a transition relation. We write $s \xrightarrow{a}{}^S s'$ when $(s,a,s') \in \rightarrow^S$ and use $i?$, $o!$ and d to range over inputs, outputs and $\mathbb{R}_{\geq 0}$, respectively.*

In what follows, we assume that any TIOTS satisfies the following conditions:

- time determinism: whenever $s \xrightarrow{d}{}^S s'$ and $s \xrightarrow{d}{}^S s''$ then $s' = s''$
- time reflexivity: $s \xrightarrow{0}{}^S s$ for all $s \in St^S$
- time additivity: for all $s, s'' \in St^S$ and all $d_1, d_2 \in \mathbb{R}_{\geq 0}$ we have $s \xrightarrow{d_1+d_2}{}^S s''$ iff $s \xrightarrow{d_1}{}^S s'$ and $s' \xrightarrow{d_2}{}^S s''$ for an $s' \in St^S$

A *run* ρ of a TIOTS S from its state s_1 is a sequence $s_1 \xrightarrow{a_1}{}^S s_2 \xrightarrow{a_2}{}^S \dots \xrightarrow{a_n}{}^S s_{n+1}$ such that for all $1 \leq i \leq n$, $s_i \xrightarrow{a_i}{}^S s_{i+1}$ with $a_i \in \Sigma^S \cup \mathbb{R}_{\geq 0}$. We write $\mathsf{Runs}(s_1, S)$ for the set of runs of S starting in s_1 and $\mathsf{Runs}(S)$ for $\mathsf{Runs}(s_0, S)$. We write $\mathsf{States}(\rho)$ for the set of states reached in ρ, and if ρ is finite $\mathsf{last}(\rho)$ is the last state occurring in ρ.

A TIOTS S is *deterministic* iff $\forall a \in \Sigma^S \cup \mathbb{R}_{\geq 0}$, whenever $s \xrightarrow{a}{}^S s'$ and $s \xrightarrow{a}{}^S s''$, then $s' = s''$. It is *input-enabled* iff each of its states $s \in St^S$ is input-enabled: $\forall i? \in \Sigma_i^S . \exists s' \in St^S . s \xrightarrow{i?}{}^S s'$. It is *output urgent* iff $\forall s, s', s'' \in St^S$ if $s \xrightarrow{o!}{}^S s'$ and $s \xrightarrow{d}{}^S s''$ then $d = 0$. Finally, S verifies the *independent progress* condition iff either $(\forall d \geq 0 . s \xrightarrow{d}{}^S)$ or $(\exists d \in \mathbb{R}_{\geq 0} . \exists o! \in \Sigma_o^S . s \xrightarrow{d}{}^S s'$ and $s' \xrightarrow{o!}{}^S)$.

TIOTS are syntactically represented by *Timed I/O Automata (TIOA)*. Let *Clk* be a finite set of *clocks*. A *clock valuation* over *Clk* is a mapping $Clk \mapsto \mathbb{R}_{\geq 0}$ (thus $\mathbb{R}_{\geq 0}^{Clk}$). Given a valuation u and $d \in \mathbb{R}_{\geq 0}$, we

write $u+d$ for the valuation in which for each clock $x \in Clk$ we have $(u+d)(x) = u(x)+d$. For $\lambda \subseteq Clk$, we write $u[\lambda]$ for a valuation agreeing with u on clocks in $Clk \setminus \lambda$, and mapping to 0 the clocks in λ.

Let $\Phi(Clk)$ denote all *clock constraints* φ generated by the grammar $\varphi ::= x \prec k \mid x-y \prec k \mid \varphi \wedge \varphi$, where $k \in \mathbb{Q}, x, y \in Clk$ and $\prec \in \{<, \leq, >, \geq\}$. For $\varphi \in \Phi(Clk)$ and $u \in \mathbb{R}^{Clk}_{\geq 0}$, we write $u \models \varphi$ if u satisfies φ. Let $[\![\varphi]\!]$ denote the set of valuations $\{u \in \mathbb{R}^{Clk}_{\geq 0} \mid u \models \varphi\}$. A subset $Z \subseteq \mathbb{R}^{Clk}_{\geq 0}$ is a *zone* if $Z = [\![\varphi]\!]$ for some $\varphi \in \Phi(Clk)$.

Definition 2 *A* Timed I/O Automaton *is a tuple* $\mathsf{A} = (Loc, q_0, Clk, E, Act, Inv)$*, where Loc is a finite set of* locations*,* $q_0 \in Loc$ *is the* initial location*, Clk is a finite set of* clocks*,* $E \subseteq Loc \times Act \times \Phi(Clk) \times 2^{Clk} \times Loc$ *is a set of* edges*,* $Act = Act_{\mathrm{i}} \oplus Act_{\mathrm{o}}$ *is a finite set of* actions*, partitioned into* inputs *(Act_{i}) and* outputs *(Act_{o}),* $Inv : Loc \mapsto \Phi(Clk)$ *is a set of location* invariants*.*

We assume all TIOA include a universal location*, denoted* l_{u}*, that accepts every input and can produce every output at any time.*

The semantics of a TIOA $\mathsf{A} = (Loc, q_0, Clk, E, Act, Inv)$ is a TIOTS $[\![\mathsf{A}]\!]_{\mathrm{sem}} = (Loc \times \mathbb{R}^{Clk}_{\geq 0}, (q_0, \mathbf{0}), Act, \rightarrow)$, where $\mathbf{0}$ is a constant function mapping all clocks to zero, and $\rightarrow$ is the largest transition relation generated by the following rules:

- Each edge $(q, a, \varphi, \lambda, q') \in E$ gives rise to $(q,u) \xrightarrow{a} (q', u')$ for each clock valuation $u \in \mathbb{R}^{Clk}_{\geq 0}$ such that $u \models \varphi$ and $u' = u[\lambda \mapsto 0]$ and $u' \models Inv(q')$.
- Each location $q \in Loc$ with a valuation $u \in \mathbb{R}^{Clk}_{\geq 0}$ gives rise to a transition $(q,u) \xrightarrow{d} (q, u+d)$ for each delay $d \in \mathbb{R}_{\geq 0}$ such that $u + d \models Inv(q)$.

Let X be a set of states in $[\![\mathsf{A}]\!]_{\mathrm{sem}}$ and let $a \in Act$. The a-successors and a-predecessors of X are defined respectively by:

$$\mathsf{Post}_a(X) = \{(q', u') \mid \exists (q,u) \in X . (q,u) \xrightarrow{a} (q', u')\}$$
$$\mathsf{Pred}_a(X) = \{(q, u) \mid \exists (q', u') \in X . (q,u) \xrightarrow{a} (q', u')\}$$

The timed successors and timed predecessors of X are respectively defined by:

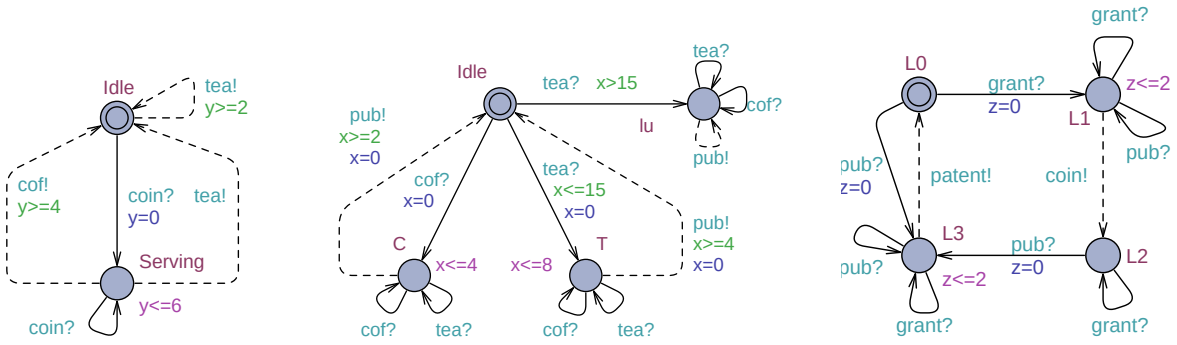
$$X^{\nearrow} = \{(q, u+d) \mid (q,u) \in X, d \in \mathbb{R}_{\geq 0}\}$$
$$X_{\swarrow} = \{(q, u-d) \mid (q,u) \in X, d \in \mathbb{R}_{\geq 0}\}$$

Additionally, we defined the safe timed predecessors of X *w.r.t* states Y, that are the timed predecessors of X that avoids the states of Y along the path:

$$\mathsf{Pred}_t(X, Y) = \{(q,u) \mid \exists d \in \mathbb{R}_{\geq 0} . (q,u) \xrightarrow{d} (q, u+d) \text{ and } (q, u+d) \in X \text{and } \forall d' \in [0, d] . (q, u+d') \notin Y\}$$

Symbolic Abstractions Since TIOTSs are infinite size they cannot be directly manipulated by computations. Usually *symbolic representations*, such as *region graphs* [5] or *zone graphs*, are used as data structures that finitely represent semantics of TIOAs. We denote by $X = (q, Z)$ a *symbolic state*, where $q \in Loc$ and $Z \subseteq \mathbb{R}^{Clk}_{\geq 0}$ is a zone. The *zone graph* is $\mathsf{G}_{\mathsf{A}} = (\mathcal{Z}_{\mathsf{A}}, X_0, \rightarrow)$, where $\mathcal{Z}_{\mathsf{A}}$ is the set of reachable zones. The initial state is defined by $X_0 = \{(q_0, \mathbf{0})\}^{\nearrow} \cap [\![Inv(q_0)]\!]$. For $a \in Act$, $(q, Z) \xrightarrow{a} Z (q', Z')$ if $(q, a, \varphi, \lambda, q') \in E$ and $Z' = ((Z \cap [\![\varphi]\!])[\lambda])^{\nearrow} \cap [\![Inv(q')]\!]$.

Example Figure 2 presents three small examples of TIOAs, that specifies the behaviour of a university composed by a coffee machine (Fig. 2a), a researcher (Fig. 2b) and an administration (Fig. 2c).



(a) Coffee machine (b) Researcher (c) Administration

Figure 2: Timed specifications with timed I/O automata

Timed Games TIOAs are interpreted as two-player real-time games between the *output player* (the component) and the *input player* (the environment). The input plays with actions in Act_{i} and the output plays with actions in Act_{o}. A strategy for a player is a function that defines her move at a certain time (either delaying or playing a controllable action). A strategy is called *memoryless* if the next move depends solely on the current state. We only consider memoryless strategies, as these suffice for safety games [1]. For simplicity, we only define strategies for the output player (i.e. output is the verifier). Definitions for the input player are obtained symmetrically.

Definition 3 *A memoryless* strategy f_o *for the output player on the TIOA* A *is a partial function* $St^{[\![\mathsf{A}]\!]_{\mathrm{sem}}} \mapsto Act_{\mathrm{o}} \cup \{\mathsf{delay}\}$*, such that*

- *Whenever* $f_o(s) \in Act_{\mathrm{o}}$ *then* $s \xrightarrow{f_o(s)} s'$ *for some* s'.
- *Whenever* $f_o(s) = \mathsf{delay}$ *then* $s \xrightarrow{d} s''$ *for some* $d > 0$ *and state* s''*, and* $f_o(s'') = \mathsf{delay}$.

The game proceeds as a concurrent game between the two player, each proposing its own strategy. The restricted behaviour of the game defines the *outcome* of the strategies.

Definition 4 *Let* A *be a TIOA,* f_o *and* f_i *be two strategies over* A *for the output and input player, respectively, and* s *be a state of* $[\![\mathsf{A}]\!]_{\mathrm{sem}}$*.* $\mathsf{Outcome}(s, f_o, f_i)$ *is the subset of* $\mathsf{Runs}(s, [\![\mathsf{A}]\!]_{\mathrm{sem}})$ *defined inductively by:*

- $s \in \mathsf{Outcome}(s, f_o, f_i)$,
- *if* $\rho \in \mathsf{Outcome}(s, f_o, f_i)$*, then* $\rho' = \rho \xrightarrow{a} s' \in \mathsf{Outcome}(s, f_o, f_i)$ *if* $\rho' \in \mathsf{Runs}(s, [\![\mathsf{A}]\!]_{\mathrm{sem}})$ *and one the following conditions hold:*
  1. $a \in Act_o$ *and* $f_o(\mathsf{last}(\rho)) = a$,
  2. $a \in Act_i$ *and* $f_i(\mathsf{last}(\rho)) = a$,
  3. $a \in \mathbb{R}_{\geq 0}$ *and* $\forall d \in [0, a[\, \exists s''.\, \mathsf{last}(\rho) \xrightarrow{d} s''$ *and* $\forall k \in \{o, i\}\ f_k(s'') = \mathsf{delay}$.
- $\rho \in \mathsf{Outcome}(s, f_o, f_i)$ *if* ρ *infinite and all its finite prefixes are in* $\mathsf{Outcome}(s, f_o, f_i)$.

A *winning condition* for a player in the TIOA A is a subset of $\mathsf{Runs}([\![\mathsf{A}]\!]_{\mathrm{sem}})$. In safety games the winning condition is to avoid a set Bad of "bad" states. Formally, the winning condition is $W^{\mathrm{o}}(\mathsf{Bad}) = \{\rho \in \mathsf{Runs}([\![\mathsf{A}]\!]_{\mathrm{sem}}) \mid \mathsf{States}(\rho) \cap \mathsf{Bad} = \emptyset\}$. A strategy f_o for output is a *winning strategy* from state s

if and only if, for all strategy f_i of input, $\mathsf{Outcome}_o(s, f_o, f_i) \subseteq W^{\mathrm{o}}(\mathsf{Bad})$. On the contrary, a strategy f_i for input is a *spoiling strategy* of f_o if and only if $\mathsf{Outcome}(s, f_o, f_i) \not\subseteq W^{\mathrm{o}}(\mathsf{Bad})$. A state s is winning for output if there exists a winning strategy from s. The game $(\mathsf{A}, W^{\mathrm{o}}(\mathsf{Bad}))$ is winning if and only if the initial state is winning. Solving this game is decidable [21, 12, 15]. We only consider safety games in this paper, and without lost of generality we assume these "bad" states correspond to a set of entirely "bad" locations.

Symbolic Timed Games: It is proved in [1] that timed games can be solved using region strategies, where the players only need to remember the sequence of regions, instead of the sequence of states used in Definition 3. Consequently timed games can be solved through symbolic computations performed on the symbolic graph (either the region graph or the zone graph) using for instance the algorithm presented in [12]. To represent these strategies we defined *symbolic strategies* which apply on symbolic states:

Definition 5 *A* symbolic strategy F_o *for the output player on the symbolic graph* $\mathsf{G}_\mathsf{A} = (\mathcal{Z}_\mathsf{A}, X_0, \rightarrow)$*, is a function* $\mathcal{Z} \mapsto Act_{\mathrm{o}} \cup \{\mathsf{delay}\}$*, where* $\mathcal{Z}$ *is a partition of the reachable states that refines* $\mathcal{Z}_\mathsf{A}$*, such that whenever* $F_o((q,Z)) \in Act_{\mathrm{o}}$ *then* $\forall u \in Z.\,(q,u) \xrightarrow{F_o((q,Z))} (q',u')$ *for some* (q',u')*.*

We remark that a symbolic strategy F_o corresponds to the set of strategies f_o such that whenever $F_o((q,Z)) = a$, then $\exists u \in Z.\, f_o((q,u)) = a$. For $(q,u) \in [\![\mathsf{A}]\!]_{\mathrm{sem}}$, if $\exists Z.\, u \in Z$ and $F((q,Z)) \in Act \cup \{\mathsf{delay}\}$, we define by extension $F((q,u)) = F((q,Z))$. For a symbolic state X we define the timed successors of X restricted by F by:

$$X\nearrow^F = \{(q,u+d) \mid (q,u) \in X,\, d \in \mathbb{R}_{\geq 0},\, \forall d' \in [0,d].\ F((q,u+d')) = F((q,u+d)) \vee F((q,u+d')) = \{\mathsf{delay}\}\}$$

3 Robust Timed Specifications

We summarize in this section the theory of robust timed specifications presented in [20]. It extends the theory of timed specifications based on TIOA presented in [15].

3.1 Basics of the Timed Specification Theory

In [15] specifications and implementations are both represented by TIOAs satisfying additional conditions:

Definition 6 *A* specification S *is a TIOA whose semantics* $[\![\mathsf{S}]\!]_{\mathrm{sem}}$ *is deterministic and input-enabled.*

Definition 7 *An* implementation I *is a specification whose semantics* $[\![\mathsf{I}]\!]_{\mathrm{sem}}$ *additionally verifies the output urgency and the independent progress conditions.*

In specification theories, a *refinement* relation plays a central role. It allows to compare specifications, and to relate implementations to specifications. In [15], as well as in [2, 3, 11], refinement is defined in the style of *alternating (timed) simulation*. Formally, given two specifications S and T, we say that S *refines* T, written $\mathsf{S} \leq \mathsf{T}$, if and only if $[\![\mathsf{S}]\!]_{\mathrm{sem}}$ is simulated by $[\![\mathsf{T}]\!]_{\mathrm{sem}}$.

Definition 8 *An implementation* I satisfies *a specification* S*, denoted* I **sat** S*, if and only if* $\mathsf{I} \leq \mathsf{S}$

A specification S is *consistent* if and only if there exists at least one implementation that satisfies S.

A complete specification theory includes several operators to compose specifications. The *parallel composition* of two specifications S and T (denoted $\mathsf{S} \parallel \mathsf{T}$) is defined by the product of the two TIOAs where components synchronize on common inputs/outputs. Additional operators include conjunction and quotient. Their definition can be found in [15].

The parallel composition may introduce some incompatible states in the product, *i.e.* states in which the two components cannot work together. With the input-enableness hypothesis no "model-related" errors can occurs when computing the product. However specific incompatible states can be introduced in the models, by using for instance the universal location l_{u} to specify an unpredictable behaviour of the component. A compatible environment for the two components allows to avoid these error states. We follow the optimistic approach of [2], *i.e. two specifications can be composed if there exists at least one environment in which they can work together*. Formally, given a set und of undesirable states, we say that a specification S is *useful* if there exists an environment E such that $[\![\mathsf{S} \parallel \mathsf{E}]\!]_{sem} \cap \mathsf{und} = \emptyset$. Two specifications S and T are *compatible* if and only if their product $\mathsf{S} \parallel \mathsf{T}$ is useful.

3.2 Strategies in Timed Games as Operators on Timed Specifications

The specification theory provides a game-based methodology in which winning strategies are used to synthesize implementations and compatible environments. Therefore, it determines consistency and usefulness of specifications,

In the *consistency game* the output player tries to verify a safety condition, *i.e.* avoid a set of immediate inconsistent sates $\mathsf{err}^{\mathsf{S}} \subseteq St^{[\![\mathsf{S}]\!]_{\mathrm{sem}}}$. Those are the sates that violate the independent progress condition:

$$\mathsf{err}^{\mathsf{S}} = \left\{ s \,\middle|\, (\exists d.\, s \overset{d}{\not\longrightarrow}) \text{ and } \forall d \forall o! \forall s'.\, s \overset{d}{\longrightarrow} s' \text{ implies } s' \overset{o!}{\not\longrightarrow} \right\}$$

If output has a winning strategy f_o in the timed game $(\mathsf{S}, W^{\mathrm{o}}(\mathsf{err}^{\mathsf{S}}))$, then one can synthesize from f_o an implementation I of S.

On the contrary in the *usefulness game* the input player tries to avoid the set of incompatible states. If there exists a winning strategy f_i in the game $(\mathsf{S}, W^{\mathrm{o}}(\mathsf{und}^{\mathsf{S}}))$, it provides a compatible environment for S. This allows to prove usefulness of specifications and therefore compatibility between two specifications.

3.3 Robust Implementations

An essential requirement for an implementation is to be realizable on a physical hardware, but this requires admitting small imprecisions characteristic for physical components (computer hardware, sensors and actuators). The requirement of realizability has already been linked to the robustness problem in [25] in the context of model checking. In specification theories the small deficiencies of hardware can be reflected in a strengthened satisfaction relation, which introduces small perturbations to the timing of implementation actions, before they are checked against the requirements of a specification—ensuring that the implementation satisfies the specification even if its behaviour is perturbed.

We first formalize the concept of perturbation. Let $\varphi \in \Phi(Clk)$ be a guard over the set of clocks Clk, let $x \in Clk$ and $k \in \mathbb{Q}$. The *enlarged guard* $\lceil \varphi \rceil_{\Delta}$ is constructed according to the following rules:

- Any term $x \prec k$ of φ with $\prec \in \{<, \leq\}$ is replaced by $x \prec k + \Delta$
- Any term $x \succ k$ of φ with $\succ \in \{>, \geq\}$ is replaced by $x \succ k - \Delta$

Similarly, the *restricted guard* $\lfloor \varphi \rfloor_{\Delta}$ is using the two following rules:

- Any term $x \prec k$ of φ with $\prec \in \{<, \leq\}$ is replaced by $x \prec k - \Delta$

- Any term $x \succ k$ of φ with $\succ \in \{>, \geq\}$ is replaced by $x \succ k+\Delta$.

Notice that for a for a clock valuation u and a guard φ, we have that $u \models \varphi$ implies $u \models \lceil \varphi \rceil_\Delta$, and $u \models \lfloor \varphi \rfloor_\Delta$ implies $u \models \varphi$, and $\lfloor \lceil \varphi \rceil_\Delta \rfloor_\Delta = \lceil \lfloor \varphi \rfloor_\Delta \rceil_\Delta = \varphi$.

We lift the perturbation to implementation TIOAs. Given a jitter Δ, the perturbation means a Δ-enlargement of invariants and of output edge guards. Guards on the input edges are restricted by Δ:

Definition 9 *For an implementation* $\mathsf{I} = (Loc, q_0, Clk, E, Act, Inv)$ *and* $\Delta \in \mathbb{Q}_{>0}$*, the* Δ-perturbation *of* I *is the TIOA* $\mathsf{I}_\Delta = (Loc, q_0, Clk, E', Act, Inv')$*, such that:*

- *Every edge* $(q, o!, \varphi, \lambda, q') \in E$ *is replaced by* $(q, o!, \lceil \varphi \rceil_\Delta, \lambda, q') \in E'$*,*
- *Every edge* $(q, i?, \varphi, \lambda, q') \in E$ *is replaced by* $(q, i?, \lfloor \varphi \rfloor_\Delta, \lambda, q') \in E'$*,*
- $\forall q \in Loc. Inv'(q) = \lceil Inv(q) \rceil_\Delta$*,*
- $\forall q \in Loc. \forall i? \in Act_{\mathrm{i}}$ *there exists and edge* $(q, i?, \varphi_{\mathrm{u}}, \emptyset, l_{\mathrm{u}}) \in E'$ *with* $\varphi_{\mathrm{u}} = \neg(\bigvee_{(q,i?,\varphi,\lambda,q') \in E} \lfloor \varphi \rfloor_\Delta)$.

I_Δ is not necessarily action deterministic, as output guards are enlarged. However it is input-enabled, since by construction (last case in previous definition), any input not accepted after restricting input guards is redirected to the universal location l_{u}. Also I_0 equals I.

In a similar manner, for a specification S we define $\lceil \mathsf{S} \rceil^{\mathrm{o}}_\Delta$ the TIOA where all output edges and invariants have been enlarged.

Definition 10 *An implementation* I *robustly satisfies a specification* S *for a given delay* $\Delta \in \mathbb{Q}_{\geq 0}$*, denoted* I **sat**$_\Delta$ S*, if and only if* $\mathsf{I}_\Delta \leq \mathsf{S}$

A specification is Δ-*robust consistent* if and only if it admits at least one Δ-robust implementation. A specification is Δ-*robust useful* is there exists an environment E, such that $\lceil \mathsf{E} \rceil^{\mathrm{o}}_\Delta \parallel \mathsf{S}$ avoids the errors states $\mathsf{und}^{\mathsf{S}}$. As previously two specifications S and T are Δ-*robust compatible* if and only if their composition is Δ-robust useful. The next property shows that robustness is monotonic for different values of the delay:

Property 1 (Monotonicity) *Given two delays* $0 < \Delta_1 \leq \Delta_2$ *and an implementation* I*:* $\mathsf{I} \leq \mathsf{I}_{\Delta_1} \leq \mathsf{I}_{\Delta_2}$ *Therefore, if a specification* S *is* Δ_2*-robust consistent, then* S *is also* Δ_1*-robust consistent. Moreover if* S *is* Δ_2*-robust useful, then* S *is* Δ_1*-robust useful.*

3.4 Robust Timed Games for Timed Specifications

Robust timed games add a robustness objective to safety games. They can be used to verify robust consistency and robust compatibility, as it was done in the non-robust cases. We have presented in [20] a notion of robust strategies for timed games, and we show how to synthesize robust implementations and robust environments from these strategies. We finally give a construction of a robust game automaton, whose original idea comes from [13], that transforms the original game. It is shown that finding strategies in this automaton, using classical timed games algorithms, permits to synthesize robust strategies in the original game. In this paper we always use with this construction to solve robust timed games. Therefore we only recall its definition below:

Definition 11 *Let* $(\mathsf{A}, W^{\mathrm{o}}(\mathsf{Bad}))$ *be a timed game, where* $\mathsf{A} = (Loc, q_0, Clk, E, Act, Inv)$ *and* $\mathsf{Bad} \in Loc$*, and let* $\Delta \in \mathbb{Q}_{>0}$*. The* robust game automaton $\mathsf{A}^\Delta_{\mathrm{rob}} = (\widetilde{Loc}, q_0, Clk \cup \{y\}, \widetilde{E}, Act \cup \{\mathrm{rob}\}, \widetilde{Inv})$ *uses an additional clock* y*, and additional input action* $\mathrm{rob} \in Act_i$*, and is constructed according to the following rules:*

- *$Loc \subseteq \widetilde{Loc}$, and for each location $q \in Loc$ and each edge $e = (q, o!, \varphi, \lambda, q') \in E$, two locations q_e^α and q_e^β are added in $\widetilde{Loc}$. The invariant of q is unchanged; the invariants of q_e^α and q_e^β are $y \leq \Delta$.*
- *Each edge $e' = (q, i?, \varphi, \lambda, q') \in E$ gives rise to the following edges in $\widetilde{E}$:*
  $(q, i?, \varphi, \lambda, q')$, $(q_e^\alpha, i?, \varphi, \lambda, q')$ *and* $(q_e^\beta, i?, \varphi, \lambda, q')$.
- *Each edge $e = (q, o!, \varphi, \lambda, q') \in E$ gives rise to the following edges in $\widetilde{E}$:*
  $(q, o!, \varphi, \{y\}, q_e^\alpha)$, $(q_e^\alpha, o!, \{y = \Delta\}, \{y\}, q_e^\beta)$, $(q_e^\alpha, \text{rob}, \varphi, \lambda, q')$, $(q_e^\beta, \text{rob}, \varphi, \lambda, q')$,
  $(q_e^\alpha, \text{rob}, \neg\varphi, \emptyset, \mathsf{Bad})$ *and* $(q_e^\alpha, \text{rob}, \neg\varphi, \emptyset, \mathsf{Bad})$ [1]

The construction is demonstrated in Fig. 3. The ideas behind the construction are that whenever output want to fire a transition $(q, o!, \varphi_o, \lambda_o, q_1)$ in the original automaton from a state (q, u) after elapsing d time units, this takes several steps in the robust automaton:

1. Output proposes to play action $o!$ at time d with the following sequence of transitions:

$$(q,u) \xrightarrow{d-\Delta} (q, u+d-\Delta) \xrightarrow{o!} (q^\alpha, u+d-\Delta) \xrightarrow{\Delta} (q^\alpha, u+d) \xrightarrow{o!} (q^\beta, u+d)$$

   Note that this forbid output to play any action with a reaction time smaller than Δ, and consequently this forbids Zeno strategies.

2. Input can perturb this move with $d' \leq \Delta$, by choosing either a smaller delay:

$$(q^\alpha, u+d-\Delta) \xrightarrow{d'} (q^\alpha, u+d-\Delta+d') \xrightarrow{\text{rob}} (q_1, u+d-\Delta+d')$$

   or a greater delay:

$$(q^\beta, u+d) \xrightarrow{d'} (q^\beta, u+d+d') \xrightarrow{\text{rob}} (q_1, u+d+d')$$

3. At any time in locations q,q^α and q^β, the original input edge $(q, i?, \varphi_i, \lambda_i, q_1)$ is still available.
4. Output is implicitly forbidden to play a move that could not be perturbed since input will immediately win if the guard φ_o is exceeded.

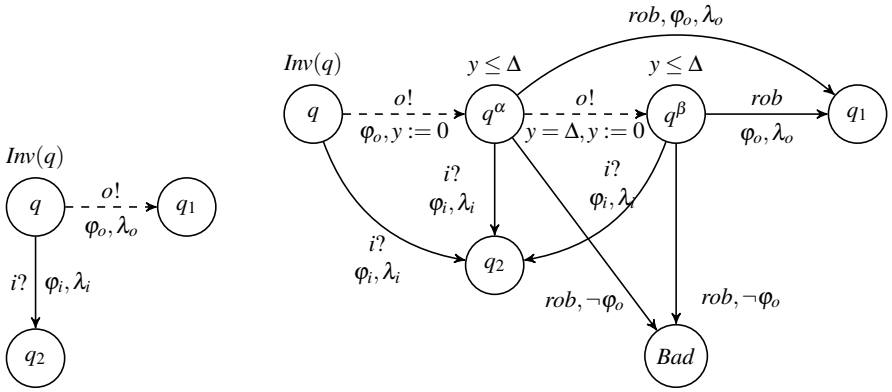
In [20], we prove that this construction is a sound technique to solve robust timed games and check robust consistency and robust compatibility.

4 Counter Strategy Refinement For Parametric Robustness

In previous section we have recalled our notions of robustness for a fixed delay. In [20] we additionally study the properties of these perturbations with respect to the different operators in the specification theory. In this paper we now consider the parametric problems, *i.e.* determining the existence of a non-null delay. More precisely due to the monotonicity properties we would like to evaluate the greatest possible value of the perturbation. The robustness problems that we consider in this section are the parametric extension of previously defined problems:

- *Robust Consistency:* Given a specification S, determine the greatest value of Δ such that S is Δ-robust consistent.
- *Robust Usefulness:* Given a specification S, determine the greatest value of Δ such that S is Δ-robust useful.

[1] Technically, since in a TIOA transitions guards must be convex, the last two transitions may be split into several copies, one for each convex guard in $\neg\varphi$.



(a) TIOA A (b) Robust game automaton $\mathsf{A}^{\Delta}_{\text{rob}}$

Figure 3: Construction of the robust game automaton $\mathsf{A}^{\Delta}_{\text{rob}}$ from an original automaton A.

4.1 Parametric Timed Games

When we consider Δ as a free parameter, the robust game automaton construction of Section 3 defines a Parametric Timed I/O Automata, in a similar manner as Parametric Timed Automata are defined in [6, 18]. We denote by $\Phi_\Delta(Clk)$ the set of parametric guards with parameter Δ over a set of clocks Clk. Parametric guards in $\Phi_\Delta(Clk)$ are generated by the following grammar $\varphi ::= x \prec l \mid x-y \prec l \mid \varphi \wedge \varphi$, where $x,y \in Clk$, $\prec \in \{<,\leq,>,\geq\}$ and $l = a + b*\Delta$ is a linear expression such that $a,b \in \mathbb{Q}$.

Definition 12 *A* Parametric TIOA *with parameter* Δ*, is a TIOA* A *such that guards and invariants are replaced by parametric guards.*

For a given value $\delta \in \mathbb{Q}_{\geq 0}$, we define the non-parametric game A_δ obtained by replacing each occurrence of the parameter Δ in the parametric guards of A by the value δ.

A parametric symbolic state X is a set of triple (q,u,δ), where δ is a value of the parameter Δ and (q,u) is a state in $[\![\mathsf{A}_\delta]\!]_{\text{sem}}$. Operations on symbolic states can be extended to parametric symbolic states, such that $X\nearrow^P$, $X\swarrow^P$,$\mathsf{PPost}_a(X)$, $\mathsf{PPred}_a(X)$ and $\mathsf{PPred}_t(X,Y)$ stands for the extensions of previously defined non-parametric operations. Formally:

$$
\begin{aligned}
X\nearrow^P &= \{(q,u+d,\delta) \mid (q,u,\delta) \in X, d \in \mathbb{R}_{\geq 0}\} \\
X\swarrow^P &= \{(q,u-d,\delta) \mid (q,u,\delta) \in X, d \in \mathbb{R}_{\geq 0}\} \\
\mathsf{PPost}_a(X) &= \{(q',u',\delta) \mid \exists (q,u,\delta) \in X.\, (q,u) \xrightarrow{a}_{\mathsf{A}_\delta} (q',u')\} \\
\mathsf{PPred}_a(X) &= \{(q,u,\delta) \mid \exists (q',u',\delta) \in X.\, (q,u) \xrightarrow{a}_{\mathsf{A}_\delta} (q',u')\} \\
\mathsf{PPred}_t(X,Y) &= \{(q,u,\delta) \mid \exists d \in \mathbb{R}_{\geq 0}.\, (q,u) \xrightarrow{d}_{\mathsf{A}_\delta} (q,u+d) \\
&\qquad \text{and } (q,u+d) \in X \text{ and } \forall d' \in [0,d].\, (q,u+d',\delta) \notin Y\}
\end{aligned}
$$

4.2 Parametric Robustness Evaluation

Solving the robustness problems for any value of Δ would in general require to solve a parametric timed game. This problem is undecidable as it has been shown that parametric model-checking problem is

undecidable [6]. In this paper, we propose to compute an approximation of the maximum delay perturbation. Due to the monotonicity of the robustness problems (Property 1), we can apply an iterative evaluation procedure that searches for the maximum value until it belongs within a given precision interval. This basic procedure is describe in Algorithm 1 for the parametric game $(\mathsf{A}^{\Delta}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$ for output (again it applies symmetrically to input).

Algorithm 1: Evaluation of parametric robustness

Input: $(\mathsf{A}^{\Delta}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$: parametric robust timed game,
Δ_{max}: initial maximum value,
ε: precision
Output: Δ_{good}: maximum admissible value of Δ

```
1 begin
2     Δ_good ← 0
3     Δ_bad ← Δ_max
4     while Δ_bad − Δ_good > ε do
5         (Δ_good, Δ_bad) ← RefineValues(A^Δ_rob, Δ_good, Δ_bad)
6     end
7     return Δ_good
8 end
```

The algorithm assumes that the game $(\mathsf{A}^{0}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$ is won, whereas the game $(\mathsf{A}^{\Delta_{max}}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$ is lost. It verifies two invariants: Δ_{good} stores the maximum value known to be correct for the robust game; Δ_{bad} stores the minimum value known to be incorrect with precision ε. At the heart of the algorithm the procedure `RefineValues` plays the game for a chosen value, and update the variables Δ_{good} and Δ_{bad} according to the result. Termination is ensure if each iteration reduces the length of the interval by some fixed minimum amount.

Different algorithms can be used to implement `RefineValues`. A basic method is binary search. In that case `RefineValues` chooses the middle point Δ_{mid} of the interval $[\Delta_{good}, \Delta_{bad}]$, and plays the game $(\mathsf{A}^{\Delta_{mid}}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$. According to the results, it updates either Δ_{good} or Δ_{bad}. This algorithm has several drawbacks. First, the number of games it needs to solve heavily depends on the precision parameter. Second, depending on the initial maximum value a high proportion of the games played may be winning, and in that case the complete symbolic graph of the model must be explored.

4.3 Counter Strategy Refinement

We propose an alternative method that follows the principle of *counterexample-guided abstraction refinement* [14]. In our settings, counterexamples are spoiling strategies computed when the game is lost. We analyse these strategies in order to refine the value of Δ. Using this technique only the last game is winning. The different steps are:

1. Play the game $(\mathsf{A}^{\Delta_{bad}}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$.
2. If the game is won, return the values $(\Delta_{bad}, \Delta_{bad})$.
3. Else extract a counter strategy F_i for the input player.
4. Replay F_i on the parametric game using Algorithm 2; it returns a value Δ_{min}.
5. If Δ_{min} is only an infimum and $\Delta_{bad} - \Delta_{min} > \varepsilon$, return the values $(\Delta_{good}, \Delta_{min})$.

6. Else return the values $(\Delta_{good}, \Delta_{min} - \varepsilon)$.

The goal of Algorithm 2 is to replay the spoiling strategy F_i on the parametric game and compute the maximum value of Δ such that this strategy becomes infeasible. It takes as inputs the parametric game automaton $\mathsf{A}^{\Delta}_{\text{rob}}$, the symbolic graph $(\mathfrak{T}^{\Delta_{bad}}_{\mathsf{A}}, X_0, \rightarrow)$ computed for the game $(\mathsf{A}^{\Delta_{bad}}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$, and the spoiling strategy F_i. It returns the infimum of the values Δ_{bad} such that F_i is a spoiling strategy in the game $(\mathsf{A}^{\Delta_{bad}}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$.

The algorithm is similar to the timed game algorithm proposed in [12] and implemented in the tool TIGA [8]. However only the backward analysis is applied on parametric symbolic states, starting from the "bad" locations. Additionally the algorithm only explores the states that belongs to the outcome of F_i. Since F_i is a spoiling strategy in a safety game, its outcome contains a set of finite runs that eventually reach the "bad" locations. This ensures that a backward exploration restricted to this set of finite runs will terminate. Formally, we define the outcome of symbolic spoiling strategy F_i for input. $\mathsf{Outcome}(F_i)$ is the subset of runs in the symbolic graph defined inductively by:

- $(q_0, S_0 \nearrow^{F_i}) \in \mathsf{Outcome}(F_i)$,
- if $\rho \in \mathsf{Outcome}(F_i)$ and $\mathsf{last}(\rho) = (q, Z)$, then $\rho' = \rho \rightarrow (q', Z') \in \mathsf{Outcome}(F_i)$ if $\exists (q, a, \varphi, \lambda, q') \in E$ and one of the following condition holds:
  1. either $a \in Act_i$ and $\exists Z''.F_i(Z'') = a$ and $Z' = \mathsf{Post}_a(Z \cap Z'') \nearrow^{F_i}$,
  2. or $a \in Act_o$ and $\exists Z''.F_i(Z'') = \mathsf{delay}$ and $Z' = \mathsf{Post}_a(Z \cap Z'') \nearrow^{F_i}$,

The backward exploration ends when the set of winning states $PWin[X_0]$ contains the initial state. Then, the projection $(PWin[X_0] \cap \mathbf{0})_{|\Delta}$ computes the set of all the valuations of Δ such that the strategy F_i is winning. The algorithm returns the infimum of these valuations.

5 Implementation

The specification theory described in [15] is implemented in the tool ECDAR [16]. In order to experiment the methods proposed in this paper, we have built a prototype in Python that reimplements the main functionalities of ECDAR and support the analysis of the robustness of timed specifications [23]. Inside this tool, the theory presented in Section 3 is implemented as a set of model transformations:

1. Computation of I_{Δ}, the Δ-perturbation of an implementation I for some $\Delta \in \mathbb{Q}_{\geq 0}$.
2. Computation of the robust game automaton $\mathsf{A}^{\Delta}_{\text{rob}}$.
3. In order to add rational perturbations on the models I_{Δ} and $\mathsf{A}^{\Delta}_{\text{rob}}$ the tool scales all the constants in the TIOA.
4. Finally we transform the TIOA of a specification into a specific *consistency game automaton* (resp. *usefulness game automaton*), such that all non Δ-robust consistent (resp. non Δ-robust useful) states are observed by a single location.

By combining these transformations we can check in the tool the three problems: Δ-robust satisfaction, Δ-consistency and Δ-usefulness. The algorithms used are respectively the alternating simulation algorithm presented in [12] and the on-the-fly timed games algorithm presented in [11].

To solve the parametric robustness problems we have implemented the heuristic presented in Section 4 that approximates the maximum solution through a counter strategy refinement. We have also implemented a binary search heuristic in order to compare the performances of the two approaches. In Algorithm 2, operations on parametric symbolic states are handled with the Parma Polyhedra Library

Algorithm 2: Counter strategy refinement

Input: $(\mathsf{A}^{\Delta}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$: parametric robust timed game,
$(\mathfrak{Z}^{\Delta_{new}}_{\mathsf{A}}, X_0, \rightarrow)$: symbolic graph computed for the game $(\mathsf{A}^{\Delta_{new}}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$
F_i: spoiling strategy for input in the game $(\mathsf{A}^{\Delta_{new}}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$

Output: Infimum of Δ_{bad} values such that F_i is a spoiling strategy in $(\mathsf{A}^{\Delta_{bad}}_{\mathrm{rob}}, W^{\mathrm{o}}(\mathsf{Bad}))$

1 **begin**
  `/* Initialisation */`
2 $Waiting \leftarrow \emptyset$
3 **for** $X = (q, Z) \in \mathsf{Z}_{\mathsf{A}}$ **do**
4 **if** $q \in \mathsf{Bad}$ **then**
5 $PWin[X] \leftarrow [\![Inv(q)]\!]$
6 $Waiting \leftarrow Waiting \cup \{Y \mid \exists \rho . \rho \rightarrow Y \rightarrow X \in \mathsf{Outcome}(F_i)\}$
7 **else**
8 $PWin[X] \leftarrow \emptyset$
9 **end**
10 **end**
  `/* Backward exploration */`
11 **while** $(Waiting \neq \emptyset) \wedge \mathbf{0} \notin PWin[X_0])$ **do**
12 $X = (q, Z) \leftarrow \mathsf{pop}(Waiting)$
13 $PBad^* \leftarrow \neg[\![Inv(q)]\!] \cup (\bigcup_X \xrightarrow{a \in Act_i}_Y \mathsf{PPred}_a(Win[Y]))$
14 $PGood^* \leftarrow \bigcup_X \xrightarrow{a \in Act_o}_Y \mathsf{PPred}_a([\![Inv(Y)]\!] \setminus PWin[Y])$
15 $PWin[X] \leftarrow \mathsf{PPred}_t(PBad^*, PGood^* \setminus PBad^*)$
16 $Waiting \leftarrow Waiting \cup \{Y \mid \exists \rho . \rho \rightarrow Y \rightarrow X \in \mathsf{Outcome}(F_i)\}$
17 **end**
18 **return** $\mathsf{Minimize}((PWin[X_0] \cap \mathbf{0})_{|\Delta})$
19 **end**

[7]. We shall remark that using polyhedra increases the complexity of computations compared to Difference Bound Matrices (DBMs), but this is necessary due to the form of the parametric constraints that are beyond the scope of classical DBMs. This not so much a problem in our approach as parametric analysis is limited to spoiling strategies whose size is kept as small as possible. Nevertheless an interesting improvement can be to use Parametric DBMs as presented in [18].

6 Experiments

We evaluate the performances of the tool to solve the parametric robustness problems on two academic examples. We compare in these experiments the Counter strategy Refinement (CR) approach with the Binary Search (BS) method. We presents benchmarks results for different values of the initial parameters Δ_{max} and ε.

	Game size		$\Delta_{max} = 8$ $\varepsilon = 0.1$		$\Delta_{max} = 6$ $\varepsilon = 0.1$		$\Delta_{max} = 8$ $\varepsilon = 0.01$		$\Delta_{max} = 6$ $\varepsilon = 0.01$	
Model	loc.	trans.	CR	BS	CR	BS	CR	BS	CR	BS
M	9	21	119ms	314ms	119ms	262ms	119ms	438ms	119ms	437ms
R	11	27	188ms	303ms	188ms	299ms	188ms	419ms	188ms	523ms
A	9	22	133ms	316ms	133ms	287ms	133ms	441ms	133ms	483ms
M $\parallel$ **A**	41	158	10.1s	10.1s	10.1s	9.6s	10.4s	17.5s	10.4s	17.6s
R $\parallel$ **A**	48	201	14.1s	12.1s	12.5s	11s	14.1s	19.6s	12.5s	19.4s
M $\parallel$ **R**	44	152	10s	15.5s	9.81s	15.8s	10.3s	22.9s	9.78s	29.2s
M $\parallel$ **R** $\parallel$ **A**	180	803	54.4s	56.3s	54.6s	112s	55s	58.8s	55.7s	216s

Table 1: Robust consistency of the university specifications

	Game size		$\Delta_{max} = 8$ $\varepsilon = 0.1$		$\Delta_{max} = 6$ $\varepsilon = 0.1$		$\Delta_{max} = 8$ $\varepsilon = 0.01$		$\Delta_{max} = 6$ $\varepsilon = 0.01$	
Model	loc.	trans.	CR	BS	CR	BS	CR	BS	CR	BS
M $\parallel$ **R**	21	90	2.64s	4.34s	1.72s	4.02s	2.64s	5.5s	1.72s	5.45s
M $\parallel$ **R** $\parallel$ **A**	75	399	48s	65s	42.7s	74.2s	48.2s	78.1s	42.9s	120s

Table 2: Robust compatibility between the university specifications

6.1 Specification of a university

The toy examples featured in this paper are extracted from [15]. They describe the overall specification of a university, composed by three specifications: the coffee machine (M) of Fif. 2a, the researcher (R) of Fig. 2b, and the administration (A) of Fif. 2c. We study the robust consistency and the robust compatibility of these specifications and their parallel composition. The results are presented in Tables 1 and 2. The column *game size* displays the size of the robust game automaton used in the analysis in terms of locations (*loc.*) and transitions (*trans.*). The next columns display the time spent to compute the maximum perturbation with different initial conditions. The analysis of these results first shows that the Counter strategy Refinement method is almost independent from the two initial parameters Δ_{max} and ε. This is not the case for Binary Search: the precision ε influences the number of games that must be solved, and the choice of Δ_{max} change the proportion of games that are winning. Comparing the results of the two methods shows that for most of the cases, especially the more complex one, the Counter strategy Refinement approach is more efficient.

6.2 Specification of a Milner Scheduler

The second experiment studies a real-time version of Milner's scheduler previously introduced in [16]. The model consists in a ring of N nodes. Each nodes receives a start signal from the previous node to perform some work and in the mean time forward the token to the next node within a given time interval. We check the robust consistency of this model for different values of N and different initial parameters. The results are displayed in Table 3. Like in previous experiment the results show that the Counter strategy Refinement method is independent form the initial conditions and in general more efficient than Binary Search.

	Game size		$\Delta_{max} = 30$ $\varepsilon = 0.5$		$\Delta_{max} = 31$ $\varepsilon = 0.5$		$\Delta_{max} = 30$ $\varepsilon = 0.1$		$\Delta_{max} = 31$ $\varepsilon = 0.1$	
Model	loc.	trans.	CR	BS	CR	BS	CR	BS	CR	BS
1 Node	13	35	0.97s	0.68s	1.09s	0.72s	0.97s	1.03s	1.09s	1.09s
2 Nodes	81	344	10.7s	10.3s	11.2s	12.6s	10.5s	15.8s	11.1s	19.4s
3 Nodes	449	2640	1m58	2m25	2m06	2m26	1m57	3m39	2m05	3m45
4 Nodes	2305	17152	17m38	24m12	17m38	27m46	17m41	37m57	17m37	41m50

Table 3: Robust consistency of Milner's scheduler nodes

6.3 Interpretation

The performances of the Binary Search method depends on the number of games that are solved and on the outcome of these games. Games that are winning (or games that are losing but with a value of Δ close to the optimum value) are harder to solve, since in these cases the (almost) complete symbolic state space must be explored. Reducing the precision parameter ε implies that more games must be solved close to the optimum value, and therefore it increases the time of analysis. Moreover, changing, even slightly, the initial maximum value Δ_{max} may change the number of games, but most important the outcome of these games, and therefore the proportion of winning games. For instance in the last experiment, the expected result is 7.5. With an initial value of 30 the bisections performed by the Binary Search method arbitrarily imply that only 1 game is winning out of 9 (for $\varepsilon = 0.1$). With 31 this proportion is 6 out of 9, which increases the complexity of the analysis.

With the Counter strategy Refinement approach proposed in this paper only losing games are played until one is winning. The choice of Δ_{max} modifies the number of games that are solved, but in general the first games for large values of Δ are easily solved. Consequently, the choice of Δ_{max} shows in the experiments almost no impact on the performances. With the parametric approach the parameter ε is only used when the value Δ_{min} computed by the refinement process is the minimum of the bad values. In that case the next iteration plays the game with the value $\Delta_{min} - \varepsilon$. The experiments shows this has no impact on the performances.

7 Conclusion

We have studied the parametric robustness problems for timed specifications. This works is based on the theory of timed specifications of [15]. It extends the theory of robust specifications of [20], which was limited to fix values for the delays. More precisely, we evaluate through approximation techniques the maximum imprecision allowed by specifications. To this end, we propose a counterexample refinement approach that analyses spoiling strategies in timed games.

This technique has been implemented in a prototype tool and its performances have been evaluated during two experiments. The results show that our counterexample refinement technique offers in most cases better and more robust (*w.r.t* initial conditions) performances than the binary search technique.

In a future version of our tool, we would like to apply the counterexample refinement approach to the alternating simulation game, in order to solve the parametric satisfaction problem for an existing implementation. We will also try to improve the performances; in particular for analysing parametric symbolic states. An interesting approach could be to replace polyhedra by parametric DBMs.

References

[1] Luca de Alfaro, Marco Faella, Thomas Henzinger, Rupak Majumdar & Mariëlle Stoelinga (2003): *The Element of Surprise in Timed Games*. In: *CONCUR*, *LNCS* 2761, Springer, pp. 144–158, doi:10.1007/978-3-540-45187-7_9.

[2] Luca de Alfaro & Thomas A. Henzinger (2001): *Interface automata*. In: *ESEC / SIGSOFT FSE*, pp. 109–120, doi:10.1145/503209.503226.

[3] Luca de Alfaro & Thomas A. Henzinger (2004): *Interface-Based Design*. In: *In Engineering Theories of Software Intensive Systems, Marktoberdorf Summer School*, doi:10.1.1.77.4920.

[4] Luca de Alfaro, Thomas A. Henzinger & Mariëlle Stoelinga (2002): *Timed Interfaces*. In: *EMSOFT*, *LNCS* 2491, Springer, pp. 108–122, doi:10.1007/3-540-45828-X_9.

[5] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. *Theor. Comput. Sci.* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[6] Rajeev Alur, Thomas A. Henzinger & Moshe Y. Vardi (1993): *Parametric real-time reasoning*. In: *STOC*, pp. 592–601, doi:10.1145/167088.167242.

[7] R. Bagnara, P. M. Hill & E. Zaffanella (2008): *The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems*. *Science of Computer Programming* 72(1–2), pp. 3–21, doi:10.1016/j.scico.2007.08.001.

[8] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen & Didier Lime (2007): *UPPAAL-Tiga: Time for Playing Games!* In: *CAV*, *LNCS* 4590, Springer, pp. 121–125, doi:10.1007/978-3-540-73368-3_14.

[9] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson & Wang Yi (2011): *Developing UPPAAL over 15 years*. *Softw., Pract. Exper.* 41(2), pp. 133–142, doi:10.1002/spe.1006.

[10] Patricia Bouyer, Nicolas Markey & Ocan Sankur (2011): *Robust Model-Checking of Timed Automata via Pumping in Channel Machines*. In: *FORMATS*, *LNCS* 6919, Springer, Aalborg, Denmark, pp. 97–112, doi:10.1007/978-3-642-24310-3_8.

[11] Peter Bulychev, Thomas Chatain, Alexandre David & Kim G. Larsen (2009): *Efficient on-the-fly Algorithm for Checking Alternating Timed Simulation*. In: *FORMATS*, *LNCS* 5813, Springer, pp. 73–87, doi:10.1007/978-3-642-04368-0_8.

[12] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen & Didier Lime (2005): *Efficient On-the-Fly Algorithms for the Analysis of Timed Games*. In: *CONCUR*, *LNCS* 3653, Springer, pp. 66–80, doi:10.1007/11539452_9.

[13] Krishnendu Chatterjee, Thomas A. Henzinger & Vinayak S. Prabhu (2008): *Timed Parity Games: Complexity and Robustness*. In: *FORMATS*, *LNCS* 5215, Springer, Saint Malo, France, pp. 124–140, doi:10.1007/978-3-540-85778-5_10.

[14] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu & Helmut Veith (2000): *Counterexample-Guided Abstraction Refinement*. In: *CAV*, *LNCS* 1855, Springer, pp. 154–169, doi:10.1007/10722167_15.

[15] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman & Andrzej Wąsowski (2010): *Timed I/O automata: a complete specification theory for real-time systems*. In: *HSCC*, ACM, pp. 91–100, doi:10.1145/1755952.1755967.

[16] Alexandre David, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman & Andrzej Wąsowski (2010): *ECDAR: An Environment for Compositional Design and Analysis of Real Time Systems*. In: *ATVA*, *LNCS* 6252, Springer, Singapore, pp. 365–370, doi:10.1007/978-3-642-15643-4_29.

[17] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis & Sergio Yovine (1994): *Symbolic Model Checking for Real-Time Systems*. *Inf. Comput.* 111(2), pp. 193–244, doi:10.1006/inco.1994.1045.

[18] Thomas Hune, Judi Romijn, Mariëlle Stoelinga & Frits W. Vaandrager (2002): *Linear parametric model checking of timed automata*. *J. Log. Algebr. Program.* 52-53, pp. 183–220, doi:10.1016/S1567-8326(02)00037-1.

[19] Rémi Jaubert & Pierre-Alain Reynier (2011): *Quantitative Robustness Analysis of Flat Timed Automata*. In: *FOSSACS*, *LNCS* 6604, Springer, pp. 229–244, doi:10.1007/978-3-642-19805-2_16.

[20] Kim G. Larsen, Axel Legay, Louis-Marie Traonouez & Andrzej Wasowski (2011): *Robust Specification of Real Time Components*. In: *FORMATS*, *LNCS* 6919, Springer, Aalborg, Denmark, pp. 129–144, doi:10.1007/978-3-642-24310-3_10.

[21] Oded Maler, Amir Pnueli & Joseph Sifakis (1995): *On the Synthesis of Discrete Controllers for Timed Systems (An Extended Abstract)*. In: *STACS*, pp. 229–242, doi:10.1.1.164.8800.

[22] Anuj Puri (1998): *Dynamical properties of timed automata*. In: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, *LNCS* 1486, Springer, pp. 210–227, doi:10.1007/BFb0055349.

[23] Python implementation of ECDAR: *PyECDAR*. `https://launchpad.net/pyecdar`.

[24] Martin Wulf, Laurent Doyen, Nicolas Markey & Jean-François Raskin (2008): *Robust safety of timed automata*. *Formal Methods in System Design* 33, pp. 45–84, doi:10.1007/s10703-008-0056-7.

[25] Martin De Wulf, Laurent Doyen & Jean-François Raskin (2005): *Almost ASAP semantics: from timed models to timed implementations*. *Formal Aspects of Computing* 17(3), pp. 319–341, doi:10.1007/s00165-005-0067-8.