Robust Synthesis for Real Time Systems[☆]

Kim G. Larsen[a], Axel Legay[b], Louis-Marie Traonouez[a,c,*], Andrzej Wąsowski[c]

[a] *Aalborg University, Science Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark*
[b] *IRISA / INRIA Rennes, 263 Avenue du Général Leclerc, 35042 Rennes Cedex, France*
[c] *IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark*

Abstract

Specification theories for real-time systems allow reasoning about interfaces and their implementation models, using a set of operators that includes satisfaction, refinement, logical and parallel composition. To make such theories applicable throughout the entire design process from an abstract specification to an implementation, we need to reason about the possibility to effectively implement the theoretical specifications on physical systems, despite their limited precision. In the literature, this implementation problem has been linked to the robustness problem that analyzes the consequences of introducing small perturbations into formal models.

We address this problem of robust implementations in timed specification theories. We first consider a fixed perturbation and study the robustness of timed specifications with respect to the operators of the theory. To this end we synthesize robust strategies in timed games. Finally, we consider the parametric robustness problem and propose a counter-example refinement heuristic.

Keywords: Stepwise refinement, Timed I/O automata, Timed games, Specification theory, Robustness

1. Introduction

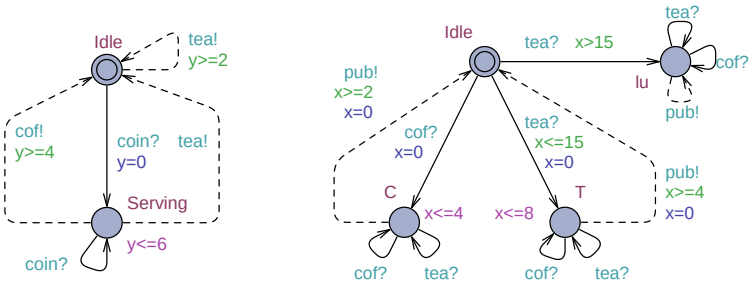
Component-based design is a software development paradigm well established in the software engineering industry. In component-based design, larger systems are built from smaller modules that depend on each other in well delimited ways described by interfaces. The use of explicit interfaces encourages creation of robust and reusable components.

Practice of component-based design is supported by a range of standardized middleware platforms such as CORBA [1], OSGi [2] or WSDL [3]. These technologies are usually not expressive enough to handle intricate correctness properties of

[☆]The research presented in this paper has been supported by MT-LAB, a VKR Centre of Excellence for the Modeling of Information Technology.

[*]Corresponding author. Phone: +45 99409857. Fax: +45 98159889.

Email addresses: kgl@cs.aau.dk (Kim G. Larsen), axel.legay@irisa.fr (Axel Legay), lmtr@cs.aau.dk (Louis-Marie Traonouez), wasowski@itu.dk (Andrzej Wąsowski)



(a) Coffee machine (b) Researcher

Figure 1: Timed specifications with timed I/O automata

safety-critical concurrent real-time software—a domain where component-based design would be particularly instrumental to address the strict legal requirements for certification [4]. To aid these needs researchers work on developing trustworthy rigorous methods for component-oriented design. In the field of concurrency verification this includes compositional design (specification theories, stepwise-refinement) and compositional model checking. Akin to algebraic specifications, *specification theories* provide a language for specifying component interfaces together with operators for combining them, such as parallel (structural) composition or conjunction (logical 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 [6]. Mature modeling and model-checking tools exist, such as Uppaal [7], that implement this technique and have been applied to numerous industrial applications.

In [8], we have proposed a specification theory for real time systems based on timed automata. A specification theory uses refinement checking instead of model-checking to support compositionality of designs and proofs from ground up. We build on an input/output extension of timed automata model to specify both models *and properties*. The set of state transitions of the timed systems is partitioned between inputs, representing actions of the environment, and outputs that represent the behavior 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. These semantics are used to define the operations of the theory, including satisfaction (can a specification be implemented), refinement (how two specifications compare), logical composition (superposition of two specifications), structural composition (combining smaller components into larger ones), and quotient (synthesizing a component in a large design).

Let us illustrate the main concepts with an example. Figure 1a displays a

specification of a coffee machine that receives an input coin? and outputs either coffee (cof!) or tea!. It can be composed with the specification of a researcher in Fig. 1b by synchronizing input with output labeled with the same channel (cof and tea). In Fig. 1b the researcher specifies that if tea arrives after 15 time units, she enters into an error state lu. We can say that if there exists an environment for these two specifications that avoids reaching this error state then the two specifications are compatible [9]. In this particular example, such an environment simply needs to provide coin? sufficiently often. In general deciding existence of safe environments is reduced to establishing whether there exists a winning strategy in the underlying timed safety game.

Besides compatibility checking, the theory of [8] is equipped with a consistency check 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 refined specification of the coffee machine in Fig. 2. This machine first proposes to make a choice of a drink, then awaits a coin, and after receiving the payment it delivers the coffee. If the payment does not arrive 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 possible due to a subtle mistake. 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.

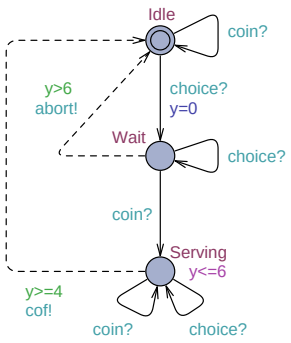


Figure 2: Non robust specification of a coffee machine

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. After all the machine does not control how much time passes between the choice of the drink and the payment. An alternative simple fix is to allow the timeout abort transition to be taken earlier — for example after 4 time units. This would guarantee at least 2 time units for brewing the coffee. Despite both corrections being quite simple, it is clear that subtle timing mistakes like this one are very hard to spot. 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. These problems appear in compositions of independent components, which may appear spuriously compatible, but their

compatibility requires infinitely precise execution platform and (or) an infinitely fast environment.

The timing precision errors in specifications are not handled in any way in idealized interface theories such as [8, 10]. These and similar issues have let to a definition of the 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. Providing a solution to this problem in the setting of timed I/O specifications is the objective of this paper. Our contributions include:

- We propose a notion of implementation of a specification that is robust with respect to a given perturbation in the delay before an action. The concept of robust implementation is lifted to a robust satisfaction operation that takes variations of timed behaviors into account when checking whether the implementation matches the requirement of the specification.

- Classical compositional design operators are lifted to the robust setting. One of the remarkable features of this new theory is that this does not require to modify the definitions of the operators themselves and that all the good properties of a specification theory (including independent implementability) are maintained.

- We propose a consistency check for robust satisfaction. This new check relies on an extension of the classical timed I/O game to the robust setting. In [11], Chatterjee et al. show that problems on robust timed games can be reduced to classical problems on an extended timed game. We modify the original construction of [11] to take the duality of inputs and outputs into account. Then, we show how our new game can be used to decide consistency in a robust setting as well as to synthesize a robust implementation from a given specification.

- Finally, we present a technique that evaluates the greatest admissible perturbations 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. This technique and the different constructions presented in the paper are implemented in prototype tool. We demonstrate its performances compared to a simple binary search technique for finding an optimal precision value.

To the best of our knowledge, this paper presents the first theory for stepwise refinement and specification of timed systems in a robust manner. While the presentation is restricted to the theory of [8], we believe that our methods work for any timed specifications. Our experience with industrial projects shows that such realistic design theories are of clear interest [12, 13, 14].

Organization of the paper. We proceed by summarizing the state of the art in Section 2 and introducing the background on Timed Specifications (Section 3). In Section 4 we introduce methods for solving robust time games that arise

in our specification theory. These methods are used in Sections 5 to reason about consistency, conjunction, parallel composition, in order to synthesize robust implementations of real time components. In Section 6 we developed a counterexample refinement technique to measure the imprecision allowed by the specifications. Finally in Section 7 we present a prototype that implements some of the functionalities of the tool ECDAR, extended with the robustness concepts presented in this paper. We demonstrate this tool on two experiments.

2. State of The Art

In the literature the robustness problem has been considered for timed automata using logical formulas as specifications (and neglecting compositional design operators). The robust semantics for timed automata with clock drifts has been introduced by Puri [15]. The problem has been linked to the implementation problem in [16], 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 [17] that proposes another robust semantics with both clock drifts and guard enlargement. Extending [15] 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. We extend the work of [16, 17] by considering compositional design operators, stepwise-refinement, and reasoning about open systems (only closed system composition were considered so far).

We solve games for consistency and compatibility using a robust controller synthesis technique inspired by Chatterjee et al. [11], who provide synthesis techniques for robust strategies in games with parity objectives. Driven by the fact that consistency and compatibility are safety games, we restrict ourselves to safety objectives, but we extend [11] by allowing negative perturbation of delays.

Our paper is also similar to the works in [18, 19] that show how one can synthesize from any timed automaton an equivalent robust automaton. We also synthesize robust components, but we start from the specification and we apply a controller synthesis methods to the specification, rather than modifying an existing implementation.

Robustness is defined in [17] as the existence of a positive value for the imprecision of a timed automata. The papers shows that this problem is decidable, but it does not show how to synthesize the value. A bound on the value is computed in [20]. Finally a quantitative analysis is performed in [21] 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 our timed specifications context, with no major restrictions on syntax of the specifications.

There is presently no alternative specification theory for timed systems with support for robustness. A preliminary version of this paper has appeared in [22].

This version differs from the short version by including proofs of theorems, by adding two entirely new sections presenting parametric methods for robustness (Sections 6–7). The algorithms of [22] merely check whether a given specification is robust with respect to a given timing precision parameter. The new methods compute the maximum value of the precision for which the specification is robustly consistent (or two specifications are robustly compatible). We also add an experimental evaluation of the performance of the proposed methods.

3. Background on Timed I/O Specifications

We now recall the definition of Timed I/O specifications [8]. We use $\mathbb{N}$ for the set of all natural numbers, $\mathbb{R}$ 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. For $x \in \mathbb{R}_{\geq 0}$, let $\lfloor x \rfloor$ denote the integer part of x and $\langle x \rangle$ denote its fractional part. Given a function f on a domain D and a subset C of D, we denote by $f_{|C}$ the restriction of f to C.

3.1. Timed I/O Transitions Systems and Timed I/O Automata

In the framework of [8], 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. Later we shall see that input represents the behaviors of the environment in which a specification is used, while output represents behaviours of the component itself.

3.1.1. Timed I/O Transitions Systems

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,*
- $\Sigma^S = \Sigma_i^S \uplus \Sigma_o^S$ *is a finite set of actions partitioned into inputs* Σ_i^S *and outputs* Σ_o^S*,*
- *and* $\rightarrow^S \colon St^S \times (\Sigma^S \cup \mathbb{R}_{\geq 0}) \times St^S$ *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 $s' \in St^S$

A *run* ρ of a TIOTS S from its state s_1 is a sequence

$$s_1 \xrightarrow{a_1}{}^S s_2 \xrightarrow{a_2}{}^S \ldots \xrightarrow{a_n}{}^S s_{n+1}$$

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. If silent actions (denoted τ) are used, they are hidden in the runs.

A TIOTS S is *deterministic* iff the action or delay fully determines the next state: $\forall a \in \Sigma^S \cup \mathbb{R}_{\geq 0}$, whenever $s \xrightarrow{a}{}^S s'$ and $s \xrightarrow{a}{}^S s''$, then $s' = s''$. It is *input-enabled* iff there is an input transition for every input action in each of its states $s \in St^S$: $\forall i? \in \Sigma_i^S.\, \exists s' \in St^S.\, s \xrightarrow{i?}{}^S s'$.

A TIOTS is *output urgent* iff whenever an output transition is possible, no further delaying is allowed:

$$\forall s, s', s'' \in St^S \text{ if } s \xrightarrow{o!}{}^S s' \text{ and } s \xrightarrow{d}{}^S s'' \text{ then } d = 0$$

Output urgency captures predictability of timing of system's reactions. Finally, we say that a TIOTS S verifies the *independent progress* condition iff for all its states s 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 $s' \xrightarrow{o!}{}^S)$ for some state s'. If S verifies independent progress, then it can always evolve using transitions it controls (delays and outputs), regardless whether the environment collaborates or not. This property will guarantee that the environment cannot block progress of time.

3.1.2. Timed I/O Automata

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 giving 0 for 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 constraint $\varphi \in \Phi(Clk)$ and $u \in \mathbb{R}_{\geq 0}^{Clk}$, we write $u \models \varphi$ if u satisfies φ. If $Z \subseteq \mathbb{R}_{\geq 0}^{Clk}$, we write $Z \models \varphi$ if $u \models \varphi$ for all $u \in Z$. We write $[\![\varphi]\!]$ to denote the set of valuations $\{u \in \mathbb{R}_{\geq 0}^{Clk} \mid u \models \varphi\}$. A subset $Z \subseteq \mathbb{R}_{\geq 0}^{Clk}$ is a *zone* if $Z = [\![\varphi]\!]$ for some $\varphi \in \Phi(Clk)$. Let $Clk' \subset Clk$ and $Z \subseteq \mathbb{R}_{\geq 0}^{Clk}$ be a zone. We define the projection of Z on the subset of clocks Clk' as $Z_{|Clk'} = \{u' \in \mathbb{R}_{\geq 0}^{Clk'} \mid \exists u \in Z. \forall x \in Clk'.\, u'(x) = u(x)\}$.

Definition 2 *A* Timed I/O Automaton *is a tuple* $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$, *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,
- $Act = Act_{\mathrm{i}} \uplus Act_{\mathrm{o}}$ *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.

Without loss of generality we assume that invariants of a location are always included in the guards of the edges that are incident with the location. We also assume that guards are satisfiable (for any guard φ the set $[\![\varphi]\!]$ is non-empty).

A *universal location*, denoted l_{u}, in a TIOA accepts every input and can produce every output at any time. The location l_{u} will be used to model an unpredictable behavior of a component.

Example 1 *Figure 1b depicts an example of a TIOA that admits two input actions* cof? *and* tea?*, and one output action* pub!*. Transitions are labeled with an action, a guard and a set of reset clocks. Input transitions are drawn with plain arrows, while output transitions are drawn with dashed arrows. Location are labeled with a name and an invariant constraint. This example contains a universal location (*lu*).*

The semantics of a TIOA $\mathcal{A}=(Loc, q_0, Clk, E, Act, Inv)$ is a TIOTS $[\![\mathcal{A}]\!]_{\mathrm{sem}} = (Loc \times \mathbb{R}_{\geq 0}^{Clk}, (q_0, \mathbf{0}), Act, \rightarrow)$, 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 $u \in \mathbb{R}_{\geq 0}^{Clk}$ such that $u \models \varphi$ and $u' = u[\lambda]$ and $u' \models Inv(q')$.
- Each location $q \in Loc$ with a valuation $u \in \mathbb{R}_{\geq 0}^{Clk}$ gives rise to a transition $(q, u) \xrightarrow{d} (q, u + d)$ for each delay $d \in \mathbb{R}_{\geq 0}$ such that $u + d \models Inv(q)$.

Example 2 *In the example of Fig. 1b a possible run starting from initial location* Idle *is*

$$(\mathsf{Idle}, (0)) \xrightarrow{\mathsf{cof?}} (\mathsf{C}, (0)) \xrightarrow{2.6} (\mathsf{C}, (2.6)) \xrightarrow{\mathsf{pub!}} (\mathsf{Idle}, (0))$$

Let X be a set of states in $[\![\mathcal{A}]\!]_{\mathrm{sem}}$. For $a \in Act$ 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 predecessors of X are respectively defined by:

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

The *safe* timed predecessors of X with respect to a set of unsafe states Y are the set of timed predecessors of X such that the states of Y are avoided along the path:

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

These operations can be implemented symbolically on zones using Difference Bound Matrices (DBMs) [23].

3.1.3. 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. A *symbolic state* is a pair (q, Z) that combines all concrete states (q, u) such that $u \in Z$, where $q \in Loc$ and $Z \subseteq \mathbb{R}^{Clk}_{\geq 0}$. Usually symbolic states are formed combining locations with special kinds of sets of valuations: *regions* and *zones*. Recall that zones are sets expressed by clock constraints in TIOAs. We now define regions.

Let M be the integer constant with the greatest absolute value among constants appearing in guards of a TIOA[1]. A *clock region* is an equivalence class of the relation $\sim$ on clock valuations such that $u \sim v$ iff the following conditions hold:

- $\forall x \in Clk$, either $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$, or $u(x) > M$ and $v(x) > M$,
- $\forall x, y \in Clk$, $\forall k \in [-M, M]$, $u(x) - u(y) \leq k$ iff $v(x) - v(y) \leq k$,
- $\forall x \in Clk$ if $u(x) \leq M$ then $\langle u(x) \rangle = 0$ iff $\langle v(x) \rangle = 0$,

We write $r\nearrow$ for the region that is the unique direct time successor of region r, if such exists. For a clock valuation u, we write $[u]$ to denote the unique region containing u.

The *region graph* of a TIOA $\mathcal{A}$ is a triple $\mathcal{G}_{\mathcal{A}} = (\mathcal{R}_{\mathcal{A}}, r_0, \rightarrow)$, where $\mathcal{R}_{\mathcal{A}} = \{(q,[u]) \mid (q,u) \in St^{[\![\mathcal{A}]\!]_{\text{sem}}}\}$ is the set of symbolic states, $r_0 = (q_0, [\mathbf{0}])$ is the initial symbolic state, and $\rightarrow \subseteq \mathcal{R}_{\mathcal{A}} \times (Act \cup \{\tau\}) \times \mathcal{R}_{\mathcal{A}}$, such that $(q,r) \xrightarrow{\tau} (q, r\nearrow)$ iff $r\nearrow \models Inv(q)$, and $(q,r) \xrightarrow{a} (q',r')$ iff $(q,u) \xrightarrow{a} (q',u')$ for some $u \in r$ and $u' \in r'$.

The *zone graph* $\mathcal{G}'_{\mathcal{A}} = (\mathcal{Z}_{\mathcal{A}}, X_0, \rightarrow)$ is defined analogously, but using zones instead of regions. It provides a coarser abstraction, in which only discrete transitions are observed. There, $\mathcal{Z}_{\mathcal{A}}$ is the set of reachable symbolic states: $(q,Z) \in \mathcal{Z}_{\mathcal{A}}$ if Z is a zone of $\mathbb{R}^{Clk}_{\geq 0}$. The initial symbolic state is defined by $X_0 = \{(q_0, \mathbf{0}\nearrow \cap [\![Inv(q_0)]\!])\}$. For action $a \in Act$ there is an edge $(q,Z) \xrightarrow{a} (q',Z')$ iff $(q,a,\varphi,\lambda,q') \in E$ with $Z' = ((Z \cap [\![\varphi]\!])[\lambda])\nearrow \cap [\![Inv(q')]\!]$.

[1] The region graph of an automaton with rational constants can be built by scaling all constants of the automaton to work only with integers.

Example 3 *The TIOA in Figure 1b is a specification of a researcher. It accepts either coffee (*cof*) or* tea *in order to produce publications (*pub*). If* tea *is served after a too long period the researcher falls into an error state, represented by the universal state* lu*.*
An implementation of this specification is presented in Figure 3. It is output urgent since it produces pub *exactly 3 time units after receiving* cof *and 6 time units after receiving* tea*. The location* Blocked *is an implementation of the universal location that never produced* pub*.*

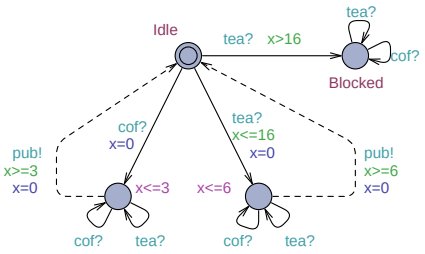


Figure 3: Implementation for a researcher

3.2. Basics of the Timed Specification Theory

In [8], timed specifications and implementations are both represented by TIOAs satisfying additional conditions:

Definition 3 *A* specification $\mathcal{S}$ *is a TIOA whose semantics* $[\![\mathcal{S}]\!]_{\text{sem}}$ *is deterministic and input-enabled. An* implementation $\mathcal{I}$ *is a specification whose semantics* $[\![\mathcal{I}]\!]_{\text{sem}}$ *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 [8], as well as in [9, 24, 25], refinement is defined in the style of alternating (timed) simulation:

Definition 4 (Refinement) *An* alternating timed simulation *between TIOTS* $S = (St^S, s_0, \Sigma^S, \rightarrow^S)$ *and* $T = (St^T, t_0, \Sigma^T, \rightarrow^T)$ *is a relation* $R \subseteq St^S \times St^T$ *such that* $(s_0, t_0) \in R$ *and for every* $(s,t) \in R$

- *If* $t \xrightarrow{i?}{}^T t'$ *for some* $t' \in St^T$*, then* $s \xrightarrow{i?}{}^S s'$ *and* $(s',t') \in R$ *for some* $s' \in St^S$
- *If* $s \xrightarrow{o!}{}^S s'$ *for some* $s' \in St^S$*, then* $t \xrightarrow{o!}{}^T t'$ *and* $(s',t') \in R$ *for some* $t' \in St^T$
- *If* $s \xrightarrow{d}{}^S s'$ *for* $d \in \mathbb{R}_{\geq 0}$*, then* $t \xrightarrow{d}{}^T t'$ *and* $(s',t') \in R$ *for some* $t' \in St^T$

We write $S \leq T$ *if there exists an alternating simulation between* S *and* T*. For two TIOAs* $\mathcal{S}$ *and* $\mathcal{T}$*, we say that* $\mathcal{S}$ refines $\mathcal{T}$*, written* $\mathcal{S} \leq \mathcal{T}$*, iff* $[\![\mathcal{S}]\!]_{\text{sem}} \leq [\![\mathcal{T}]\!]_{\text{sem}}$*.*

Definition 5 (Satisfaction) *An implementation* $\mathcal{I}$ satisfies *a specification* $\mathcal{S}$*, denoted* $\mathcal{I}$ **sat** $\mathcal{S}$*, if and only if* $[\![\mathcal{I}]\!]_{\text{sem}} \leq [\![\mathcal{S}]\!]_{\text{sem}}$*.*
We write $[\![\mathcal{S}]\!]_{\text{mod}}$ *for the set of all implementations of* $\mathcal{S}$*:*

$$[\![\mathcal{S}]\!]_{\text{mod}} = \{\mathcal{I} \mid \mathcal{I} \textbf{ sat } \mathcal{S} \quad \textit{and } \mathcal{I} \textit{ is an implementation}\}$$

The reader might find it surprising that in a robust specification theory we refrain from adjusting the refinement to account for imprecision of implementations when comparing specifications. Our basic assumption is that specifications are precise mathematical objects that are not susceptible to imprecision of execution. In contrary, implementations can behave imprecisely when executed, so in Section 4 we will introduce an extension of Def. 5 that takes this into account. It is a fortunate property of Def. 4 that we do not need to modify it in order to reason about robust implementations (Property 3 in Sect. 4).

In [8], we have reduced refinement checking to finding winning strategies in timed games. In the reminder of this section, we recall the definition of such games and show how they can be used to check consistency. Timed games also underly other operations such as conjunction, composition, and quotient [8], which will be illustrated in Sect. 5.

3.3. Timed Games for Timed I/O Specifications

TIOAs are interpreted as two-player real-time games between the *output player* (the component) and the *input player* (the environment). The input player plays with actions in Act_{i} and the output player plays with actions in Act_{o}. A strategy for a player is a function that defines her move at any given 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 [26]. 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 6 *A memoryless* strategy f_o *for the output player on the TIOA* $\mathcal{A}$ *is a partial function* $St^{[\![\mathcal{A}]\!]_{\mathrm{sem}}} \mapsto Act_{\mathrm{o}} \cup \{\mathsf{delay}\}$*, 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 behavior of the game defines the *outcome* of the strategies.

Definition 7 *Let* $\mathcal{A}$ *be a TIOA,* f_o *and* f_i *be two strategies over* $\mathcal{A}$ *for the output and input player, respectively, and* s *be a state of* $[\![\mathcal{A}]\!]_{\mathrm{sem}}$*.* $\mathsf{Outcome}(s, f_o, f_i)$ *is the subset of* $\mathsf{Runs}(s, [\![\mathcal{A}]\!]_{\mathrm{sem}})$ *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* $\rho' \in \mathsf{Runs}(s, [\![\mathcal{A}]\!]_{\mathrm{sem}})$ *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 *is infinite and all its finite prefixes are in* $\mathsf{Outcome}(s, f_o, f_i)$.

A *winning condition* for a player in the TIOA $\mathcal{A}$ is a subset of $\mathsf{Runs}(\llbracket \mathcal{A} \rrbracket_{\mathrm{sem}})$. In safety games the winning condition is to avoid a set Bad of "bad" states. Formally, the winning condition for output is $W^{\mathrm{o}}(\mathsf{Bad}) = \{\rho \in \mathsf{Runs}(\llbracket \mathcal{A} \rrbracket_{\mathrm{sem}}) \mid \mathsf{States}(\rho) \cap \mathsf{Bad} = \emptyset\}$. A strategy f_o 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 $(\mathcal{A}, W^{\mathrm{o}}(\mathsf{Bad}))$ is winning if and only if the initial state is winning. Solving this game is decidable [27, 23, 8]. 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.

Strategies in Timed Games as Operators on Timed Specifications: . We sketch how timed games can be used to establish consistency of a timed specifications.

An *immediate error* occurs in a state of a specification if the specification disallows progress of time and output transitions in a given state—such a specification will break if the environment does not send an input. For a specification $\mathcal{S}$ we define the set of immediate error states $\mathsf{err}^{\mathcal{S}} \subseteq St^{\llbracket \mathcal{S} \rrbracket_{\mathrm{sem}}}$ as:

$$\mathsf{err}^{\mathcal{S}} = \left\{ s \;\middle|\; (\exists d.\, s \xrightarrow{d} /) \text{ and } \forall d \forall o! \forall s'.\, s \xrightarrow{d} s' \text{ implies } s' \xrightarrow{\rho!} / \right\}$$

It follows that no immediate error states can occur in implementations, since they verify independent progress. In [8] we show that $\mathcal{S}$ is consistent iff there exists a winning strategy for output in the safety game $(\mathcal{S}, W^{\mathrm{o}}(\mathsf{err}^{\mathcal{S}}))$. Moreover, the maximum consistent part of $\mathcal{S}$ corresponds to the maximum wining strategy for output in this game.

Conjunction of two specifications is found as a maximal strategy for output in a safety game on the product state space to avoid immediate errors. Similarly, optimistic parallel composition of two specifications is computed as a maximum strategy for input in a safety game over the product state space; and a quotient of two specifications is found as a maximum strategy for output in another safety game. Optimistic composition means that two specifications are compatible if there exists at least one environment, in which they can avoid error states. Details can be found in [8].

4. Robust Timed I/O Specifications

We now define a robust extension of our specification theory. 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 [16] 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. This ensures that the implementation satisfies the specification even if its behavior is perturbed.

We first formalize the concept of perturbation. Let constraint $\varphi \in \Phi(X)$ be a guard over the set of clocks X. 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 $\lfloor\lceil\varphi\rceil_\Delta\rfloor_\Delta = \lceil\lfloor\varphi\rfloor_\Delta\rceil_\Delta = \varphi$.

4.1. Perturbed Implementation and Robust Timed I/O Specifications.

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 8 *For an implementation* $\mathcal{I} = (Loc, q_0, Clk, E, Act, Inv)$ *and* $\Delta \in \mathbb{Q}_{>0}$, *the* Δ-perturbation *of* $\mathcal{I}$ *is the TIOA* $\mathcal{I}_\Delta = (Loc \cup \{l_{\mathrm{u}}\}, q_0, Clk, E', Act, Inv')$, *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*

$$\varphi_{\mathrm{u}} = \neg\Big(\bigvee_{(q,i?,\varphi,\lambda,q') \in E} \lfloor\varphi\rfloor_\Delta\Big)$$

$\mathcal{I}_\Delta$ 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 $\mathcal{I}_0$ equals $\mathcal{I}$.

In essence, we weaken the constraints on output edges, and strengthen the constraints on input edges. This is consistent with the game semantics

of specifications: perturbation makes the game harder to win for the verifier. Since the gaps created by strengthening input guards are closed by edges to the universal location, the implementation becomes less predictable. If an input arrives close to the deadline, the environment cannot be certain if it will be handled precisely as specified. Enlargement of output guards has a similar effect. The environment of the specification has to be ready that outputs will arrive slightly after the deadlines.

Such considerations are out of place in classical robustness theories for model checking, but are crucial when moving to models, where input and output transitions are distinguished. For example, in [16] the authors propose a robust semantics for timed automata. Their *maximal progress assumption* is equivalent to the output urgency condition of our implementations. However, in [16] both input and output guards are increased, which is suitable for the one-player setting, but incompatible with the contravariant nature of two-player games. Such enlargement would not be monotonic with respect to the alternating refinement (Def. 4), while the perturbation of Def. 8 is monotonic.

We are now ready to define our notion of robust satisfaction:

Definition 9 *An implementation $\mathcal{I}$ robustly satisfies a specification $\mathcal{S}$ for a given delay $\Delta \in \mathbb{Q}_{\geq 0}$, denoted $\mathcal{I}$ $\mathbf{sat}_\Delta$ $\mathcal{S}$, if and only if $\mathcal{I}_\Delta \leq \mathcal{S}$.*
We write $[\![\mathcal{S}]\!]^\Delta_{\text{mod}}$ for the set of all Δ-robust implementations of $\mathcal{S}$, such that

$$[\![\mathcal{S}]\!]^\Delta_{\text{mod}} = \{\mathcal{I} \mid \mathcal{I} \ \mathbf{sat}_\Delta \ \mathcal{S} \ \wedge \ \mathcal{I} \text{ is an implementation}\}$$

Property 1 (Monotonicity) *Let $\mathcal{I}$ be an implementation and $0 \leq \Delta_1 < \Delta_2$. Then:*

$$\mathcal{I} \leq \mathcal{I}_{\Delta_1} \leq \mathcal{I}_{\Delta_2}$$

Proof 1 (Property 1) *First, observe that for any clock valuation u and guard φ, if $u \models \lceil\varphi\rceil_{\Delta_1}$ then $u \models \lceil\varphi\rceil_{\Delta_2}$, and conversely if $u \models \lfloor\varphi\rfloor_{\Delta_2}$ then $u \models \lfloor\varphi\rfloor_{\Delta_1}$. We now check the refinement between $[\![\mathcal{I}_{\Delta_1}]\!]_{\text{sem}}$ and $[\![\mathcal{I}_{\Delta_2}]\!]_{\text{sem}}$. Let $R = \{(s_1, s_2) \in St^{[\![\mathcal{I}_{\Delta_1}]\!]_{\text{sem}}} \times St^{[\![\mathcal{I}_{\Delta_2}]\!]_{\text{sem}}} \mid s_1 = (q, u) = s_2\}$ be a candidate alternating simulation relation. We prove by co-induction that R satisfies Def. 4. Consider any state (q, u) such that $(q, u)R(q, u)$.*

1. *If $(q, u) \xrightarrow{d} I_1 (q, u + d)$ for some $d \in \mathbb{R}_{\geq 0}$,*
   then $(q, u) \xrightarrow{d} I_2 (q, u + d)$, since $u + d \models \lceil Inv(q)\rceil_{\Delta_1} \Rightarrow u + d \models \lceil Inv(q)\rceil_{\Delta_2}$.
2. *If $(q, u) \xrightarrow{o!} I_1 (q', u')$, then $(q, u) \xrightarrow{o!} I_2 (q', u')$, since $u \models \lceil\varphi\rceil_{\Delta_1} \Rightarrow u \models \lceil\varphi\rceil_{\Delta_2}$ (where φ is the guard of the edge that fires $o!$) and we assumed the invariant of q' is included in the guard.*
3. *If $(q, u) \xrightarrow{i?} I_2 (q', u')$ then $(q, u) \xrightarrow{i?} I_1 (q', u')$, since $u \models \lfloor\varphi\rfloor_{\Delta_2} \Rightarrow u \models \lfloor\varphi\rfloor_{\Delta_1}$ (where φ is the guard of the edge that fires $i?$) and we assumed the invariant of q' is included in the guard.*

Clearly, both transition systems $[\![\mathcal{I}_{\Delta_1}]\!]_{\text{sem}}$ and $[\![\mathcal{I}_{\Delta_2}]\!]_{\text{sem}}$ share the same initial state s_0 and $s_0 R s_0$, which concludes the proof. The argument that $\mathcal{I} \leq \mathcal{I}_{\Delta_1}$ proceeds similarly with the same witness relation R. □

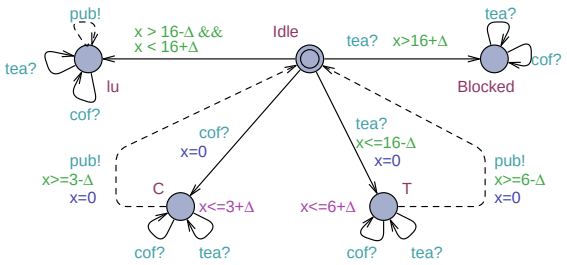


Figure 4: Δ-perturbation of the researcher implementation.

In addition, we obtain these properties by transitivity of alternating simulation:

Property 2 *Let $\mathcal{S}$ be a specification and $\Delta_1 \leq \Delta_2$. Then:*

$$[\![\mathcal{S}]\!]^{\Delta_2}_{\text{mod}} \subseteq [\![\mathcal{S}]\!]^{\Delta_1}_{\text{mod}} \subseteq [\![\mathcal{S}]\!]_{\text{mod}}$$

Property 3 *Let $\mathcal{S}$ and $\mathcal{T}$ be specifications and $0 \leq \Delta$. Then:*

$$\mathcal{S} \leq \mathcal{T} \implies [\![\mathcal{S}]\!]^{\Delta}_{\text{mod}} \subseteq [\![\mathcal{T}]\!]^{\Delta}_{\text{mod}}$$

The definition of robust satisfaction naturally induces a notion of robust consistency (implementability):

Definition 10 *Let $\mathcal{S}$ be a specification and $\Delta \in \mathbb{Q}_{>0}$, then $\mathcal{S}$ is* Δ-robust consistent *if there exists an implementation $\mathcal{I}$ such that $\mathcal{I}$ $\mathbf{sat}_\Delta$ $\mathcal{S}$.*

Like in the non-robust case, deciding consistency is reducible to solving games. But now, we will need to make the games aware of the robustness conditions. In the rest of this section, we propose a definition for such games. Then, in Section 5 we show how they can be used to perform classical operations on specifications.

Example 4 *Figure 4 presents the Δ-perturbation of the researcher implementation presented in Fig. 3. This implementation robustly satisfies the specification of Fig. 1b for any $\Delta \in]0, 1]$. However, for $\Delta = 2$ the following run is possible in the perturbed implementation:*

$$(\mathit{Idle}, (0)) \xrightarrow{\mathit{cof?}} (C, (0)) \xrightarrow{5} (C, (5))$$

but it cannot be matched by the specification because it exceeds the invariant of the location corresponding to C in Fig. 3.

4.2. Robust Timed Games for Timed I/O Specifications.

As we have seen timed specifications are interpreted as timed games. Solving games is used to analyze and synthesize real-time components. Now that we add imprecision to models, we need a notion of suitable games that can be used to synthesize robust components. Therefore we extend timed games with Δ-perturbations, and study the synthesis of robust timed strategies.

De Alfaro et al. show [26] that timed games can be solved using region strategies, where the players only need to remember the sequence of locations and clock regions, instead of the sequence of states used in Definition 6. Consequently timed games can be solved through symbolic computations performed on symbolic graphs (either the region graph or the zone graph) using for instance the algorithm presented in [23]. The following definition formalizes the notion of a *symbolic strategies* which can be represented using symbolic states only:

Definition 11 *A* symbolic strategy F *for the output player is a function* $\mathcal{Z} \mapsto Act_o \cup \{\mathsf{delay}\}$*, where* $\mathcal{Z}$ *is a set of symbolic states, such that whenever* $F((q,Z)) \in Act_o$ *then for each* $u \in Z$ *we have* $(q,u) \xrightarrow{F((q,Z))} (q',u')$ *for some* (q',u')*. A symbolic strategy for the input player is defined analogously.*

A symbolic strategy F corresponds to the set of (non-symbolic explicit) strategies f such that whenever $F((q,Z)) = a$ then $f((q,u)) = a$ for some $u \in Z$.

Syntactic outcomes.. The following construction represents the outcome of applying a symbolic strategy to a TIOA as another timed automaton. It decorates a region graph with clocks, guards and invariants. We exploit the region graph construction in the definition, but any stable partitioning of the state-space could serve this purpose, and would be more efficient in practice.

Definition 12 *Let* $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$ *be a TIOA and* F *a symbolic strategy over* $\mathcal{A}$ *for output. The TIOA* $\mathcal{A}_F = (\mathcal{R}_\mathcal{A}, (q_0,r_0), Clk, \widehat{E}, Act \cup \{\tau\}, \widehat{Inv})$ *representing the outcome of applying* F *to* $\mathcal{A}$ *is built by decorating the region graph* $\mathcal{G}_\mathcal{A} = (\mathcal{R}_\mathcal{A}, \rightarrow^G)$ *of* $\mathcal{A}$*. For each region* (q,r)*, the incident edges and the invariant are defined as follows:*

- *If* $(q,r) \xrightarrow{\tau}{}^G (q, r\nearrow)$*, then* $((q,r), \tau, r\nearrow, \emptyset, (q,r\nearrow)) \in \widehat{E}$*,*
- *If* $(q,r) \xrightarrow{i?}{}^G (q',r')$*, then for each edge* $(q, i?, \varphi, \lambda, q') \in E$*,* $((q,r), i?, \varphi, \lambda, (q',r')) \in \widehat{E}$*,*
- *If* $F((q,r)) = \mathsf{delay}$ *then* $\widehat{Inv}(q,r) = Inv(q) \wedge (r \vee r\nearrow)$*,*
- *If* $F((q,r)) = o!$*, then* $\widehat{Inv}(q,r) = r$*, and if* $(q,r) \xrightarrow{o!}{}^G (q',r')$*, then for each edge* $(q, o!, \varphi, \lambda, q') \in E$*,* $((q,r), o!, \varphi, \lambda, (q',r')) \in \widehat{E}$*.*

In a robust timed game we seek strategies that remain winning after perturbation by a delay Δ. The perturbation is defined on the syntactic outcome of the strategy, by enlarging the guards for the actions of the verifier, so that each

actions of the verifier can happen within Δ time of what the strategy originally prescribes. We write $\lceil \mathcal{A} \rceil^{\mathrm{o}}_{\Delta}$ (respectively $\lceil \mathcal{A} \rceil^{\mathrm{i}}_{\Delta}$) for the TIOA where the guards of the output (resp. input) player and the invariants have been enlarged by Δ—so every guard φ has been replaced by $\lceil \varphi \rceil_{\Delta}$ and every invariant γ by $\lceil \gamma \rceil_{\Delta}$.

Definition 13 *For a timed game* $(\mathcal{A}, W^{\mathrm{o}}(\mathsf{Bad}))$*, a symbolic strategy* F *for output is* Δ-robust winning *if it is winning when the moves of output are perturbed, i.e.*

$$\mathsf{Runs}([\![\lceil \mathcal{A}_F \rceil^{\mathrm{o}}_{\Delta}]\!]_{\mathrm{sem}}) \subseteq W^{\mathrm{o}}(\mathsf{Bad})$$

In the rest of this section we describe a technique to find these robust strategies by modifying the original game automaton.

4.3. Robust game automaton

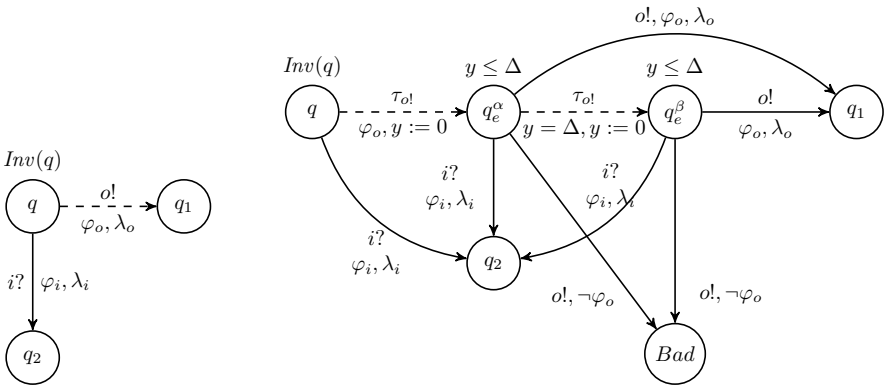
Robust timed games for a bounded delay can be reduced to classical timed games by a syntactic transformation of the game automaton [11]. Below in Def. 14, we propose an extended version of the construction presented in [11]. We admit both positive and negative perturbations of the player moves. In the original construction of [11] only delayed executions of actions were treated, but premature execution of communication may also lead to a safety violation in a specification theory, so we have to account for them. Then we show, in Theorem 1, how this construction can be used to find robust strategies as defined in Def. 13.

Let $(\mathcal{A}, W^{\mathrm{o}}(\mathsf{Bad}))$ be a timed game, where $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$ and $\mathsf{Bad} \in Loc$. We assume that all the constants in $\mathcal{A}$ are integers and we consider a perturbation $\Delta \in \mathbb{N}$.

Definition 14 *The* robust game automaton $\mathcal{A}^{\Delta}_{\mathrm{rob}} = (\widetilde{Loc}, q_0, Clk \cup \{y\}, \widetilde{E}, \widetilde{Act}, \widetilde{Inv})$ *is constructed from* $\mathcal{A}$*, with an additional clock* y*, input actions* $\widetilde{Act_{\mathrm{i}}} = Act_{\mathrm{i}} \cup Act_{\mathrm{o}}$*, and output actions* $\widetilde{Act_{\mathrm{o}}} = \{\tau_{o!} \mid o! \in Act_{\mathrm{o}}\}$*, according to the following rules:*

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

Technically speaking, 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 $\mathcal{A}$ (b) Robust game automaton $\mathcal{A}^{\Delta}_{\text{rob}}$

Figure 5: Illustration of the construction of the robust game automaton $\mathcal{A}^{\Delta}_{\text{rob}}$, from an automaton $\mathcal{A}$, with one location q and incident edges $e = (q, o!, \varphi_o, \lambda_o, q_1)$ and $(q, i?, \varphi_i, \lambda_i, q_2)$.

The construction is illustrated in Fig. 5. Intuitively, whenever the output player wants to fire a transition induced by an edge $(q, o!, \varphi_o, \lambda_o, q_1)$ in the original automaton, from a state (q, u), after elapsing d time units, in the robust automaton the input player is allowed to perturb the timing of this action. Consider the following traces on the robust game automaton that explain the construction.

1. Output proposes to play action $o!$ after a delay d with the following sequence of transitions:

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

   Note that this forbid output to play any action with a reaction time smaller than Δ. More precisely, any strategy for output found in the robust game automaton $\mathcal{A}^{\Delta}_{\text{rob}}$ will effectively correspond to a non-zeno strategy in $\mathcal{A}$.
2. Input can perturb this move with $d' \leq \Delta$, such that action $o!$ is performed with either a smaller delay:

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

   or a greater delay:

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

3. In locations q,q^{α} and q^{β}, the original input edge $(q, i?, \varphi_i, \lambda_i, q_1)$ may still be fired. So while the execution of the output $o!$ is delayed control can be intercepted by and arriving input.

4. If output reaches a state (q^α, u) or (q^β, u), with $u \not\models \varphi_o$, then input has a winning strategy with one of the following moves: $(q^\alpha, u) \xrightarrow{o!} (\mathsf{Bad}, u)$ or $(q^\beta, u) \xrightarrow{o!} (\mathsf{Bad}, u)$, that denote the late firing of action $o!$.

Let $F : \mathcal{R}_{\mathcal{A}^\Delta_{\text{rob}}} \mapsto \widetilde{Act_o} \cup \mathsf{delay}$ be a winning symbolic strategy for output in the robust game $(\mathcal{A}^\Delta_{\text{rob}}, W^o(\mathsf{Bad}))$. We construct a strategy $F_{\text{rob}} : \mathcal{R}_{\mathcal{A}} \mapsto Act_o \cup \mathsf{delay}$ for the game $(\mathcal{A}, W^o(\mathsf{Bad}))$ in the following manner. $\forall (q, r) \in \mathcal{R}_{\mathcal{A}}$,

- $F_{\text{rob}}((q, r)) = o!$ if there exists an edge $e \in E$ and a region $(q^\alpha_e . \, , \widetilde{r}) \in \mathcal{R}_{\mathcal{A}^\Delta_{\text{rob}}}$, such that $r = \widetilde{r}_{|Clk}$ and $F((q^\alpha_e . \, , \widetilde{r})) = \tau_{o!}$.
- Otherwise $F_{\text{rob}}((q, r)) = \mathsf{delay}$.

Theorem 1 *The robust game automaton is a sound construction to solve robust timed games in the following sense: if F is a winning strategy for output in the game $(\mathcal{A}^\Delta_{\text{rob}}, W^o(\mathsf{Bad}))$, then F_{rob} (constructed above) is a Δ-robust winning strategy for output in the game $(\mathcal{A}, W^o(\mathsf{Bad}))$.*

Proof 2 (Theorem 1) *We consider the automaton $\mathcal{A}_{F_{\text{rob}}}$ representing the outcome of applying F_{rob} to $\mathcal{A}$. We must prove that $\mathsf{Runs}([\![\lceil \mathcal{A}_{F_{\text{rob}}} \rceil^o_\Delta]\!]_{\text{sem}}) \subseteq W^o(\mathsf{Bad})$, where $W^o(\mathsf{Bad}) = \{\rho \in \mathsf{Runs}([\![\mathcal{A}]\!]_{\text{sem}}) \mid \mathsf{States}(\rho) \cap \mathsf{Bad} = \emptyset\}$.*

First, we map each run $\widehat{\rho}$ of $\lceil \mathcal{A}_{F_{\text{rob}}} \rceil^o_\Delta$ to a run ρ of $\mathcal{A}$. We use an induction on the length of the runs. We assume that the property holds for runs of length i: if $\widehat{\rho}_i$ is a run of $\mathsf{Runs}([\![\lceil \mathcal{A}_{F_{\text{rob}}} \rceil^o_\Delta]\!]_{\text{sem}})$ such that $\mathsf{last}(\widehat{\rho}_i) = ((q_i, r_i), u_i)$, then there exists a run ρ_i in $\mathsf{Runs}([\![\mathcal{A}]\!]_{\text{sem}})$, such that $\mathsf{last}(\rho_i) = (q_i, u_i)$.

Let $\widehat{\rho}_{i+1} = \widehat{\rho}_i \xrightarrow{a} ((q_{i+1}, r_{i+1}), u_{i+1})$. We prove the inductive step, splitting into cases:

1. *If $a \in Act_i$, then there exists an edge $\widehat{e} = ((q_i, r_i), a, \varphi, \lambda, (q_{i+1}, r_{i+1}))$ in $\mathcal{A}_{F_{\text{rob}}}$. By construction, there also exists an edge $e = (q_i, a, \varphi, \lambda, q_{i+1})$ in $\mathcal{A}$. Since $\widehat{e}$ is firable, $u_i \models \varphi$, and therefore e and $\widetilde{e}$ are also firable. So $\rho_{i+1} = \rho_i \xrightarrow{a} (q_{i+1}, u_{i+1})$ is a run of $\mathcal{A}$.*
2. *If $a \in Act_o$, then there exists $\widehat{e} = ((q_i, r_i), a, \varphi, \lambda, (q_{i+1}, r_{i+1}))$ in $\mathcal{A}_{F_{\text{rob}}}$ and by construction, $e = (q_i, a, \varphi, \lambda, q_{i+1})$ in $\mathcal{A}$. Since $\widehat{e}$ is firable, $u_i \models \lceil \varphi \rceil_\Delta$ and $u_i \models \lceil r_i \rceil_\Delta$ that is the enlarged invariant of (q_i, r_i). By construction also $F_{\text{rob}}((q_i, r_i)) = a$, which implies that there exists $\widetilde{r}_i$ such that $r_i = \widetilde{r}_{i|Clk}$ and $F((q^\alpha_i, \widetilde{r}_i)) = a$. Since $u_i \models \lceil r_i \rceil_\Delta$, $\exists \delta \in [\text{-}\Delta, \Delta]. \, u_i + \delta \models r_i$. Let $u_i = (u_{i1}, u_{i2}, \ldots, u_{in})$, then $u_i + \delta = (u_{i1} + \delta, u_{i2} + \delta, \ldots, u_{in} + \delta)$. By definition of the projection this implies that $\widetilde{u}_{i\delta} = (u_{i1} + \delta, u_{i2} + \delta, \ldots, u_{in} + \delta, \Delta) \in \widetilde{r}_i$. Now in the automaton $\mathcal{A}^\Delta_{\text{rob}}$, F is a winning strategy, which implies that $\forall \delta' \in [\text{-}\Delta, \Delta]. \, \widetilde{u}_{i\delta} + \delta' \models \varphi$ (otherwise input as a spoiling strategy). This proves that $u_i \models \varphi$ and therefore that e is firable. So $\rho_{i+1} = \rho_i \xrightarrow{a} (q_{i+1}, u_{i+1})$ is a run of $\mathcal{A}$.*
3. *If $a \in \mathbb{R}_{\geq 0}$, either the strategy F_{rob} prescribes that output can delay infinitely in $((q_i, r_i), \widehat{u}_i)$. This implies that $Inv(q_i)$ is unbounded, and that proves immediately that $\rho_{i+1} \in \mathsf{Runs}([\![\mathcal{A}]\!]_{\text{sem}})$.*

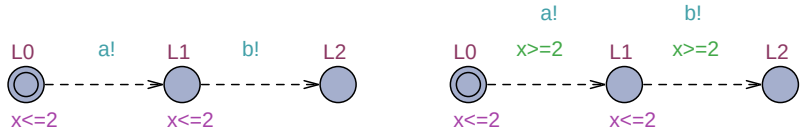
Otherwise output has a strategy that eventually performs an action. This is represented by the following sequence of edges in $\mathcal{A}_{F_{\text{rob}}}$*:* $(q_i, r_i) \xrightarrow{\tau} (q_i, r_{i1}) \xrightarrow{\tau} (q_i, r_{i2}) \dots (q_i, r_{in}) \xrightarrow{o!} (q'_i, r'_i)$*. We consider that* a *is the maximum delay firable from* $\widehat{\rho}_i$*, thus* $\widehat{\rho}_{i+1} = \widehat{\rho}_i \xrightarrow{a_1} ((q_i, r_{i1}), u_i + a_1) \xrightarrow{\tau} ((q_i, r_{i2}), u_i + a_1) \xrightarrow{a_2} ((q_i, r_{i2}), u_i + a_1 + a_2) \xrightarrow{\tau} \dots ((q_i, r_{in}), u_i + a) \xrightarrow{o!} ((q'_i, r'_i), u'_i)$*, such that* $a = \sum_{j=1}^{n} a_j$*. Then* $u_i + a \models \lceil r_{in} \rceil_\Delta$*. As in the previous case we can prove that* $u_i + a \models \varphi$*, and we assume that* φ *contains the location invariant. This proves that* $\rho_{i+1} \in \mathsf{Runs}([\![\mathcal{A}]\!]_{\text{sem}})$*.*

We now prove that all the states in $\lceil \mathcal{A}_{F_{\text{rob}}} \rceil^o_\Delta$ *are safe, by mapping the runs of* $\mathcal{A}_{F_{\text{rob}}}$ *with the ones of* $\mathcal{A}^\Delta_{\text{rob}}$*. We use another induction on the length on the run. Let assume that* $\widehat{\rho}_i \in \mathsf{Runs}([\![\lceil \mathcal{A}_{F_{\text{rob}}} \rceil^o_\Delta]\!]_{\text{sem}})$ *is mapped to a run* $\widetilde{\rho}_i \in \mathsf{Runs}([\![\mathcal{A}^\Delta_{\text{rob}}]\!]_{\text{sem}})$*, such that if* $\mathsf{last}(\widehat{\rho}_i) = ((q_i, r_i), u_i)$*, then* $\mathsf{last}(\widetilde{\rho}_i) = (q_i, \widetilde{u}_i)$*,* $u_i = \widetilde{u}_{i|Clk}$*, and* $(q_i, \widetilde{u}_i)$ *is a winning state for the strategy* F*.*

1. *Either in* $((q_i, r_i), u_i)$ *the strategy* F_{rob} *allows delaying infinitely, and consequently the strategy* F *allows delaying infinitely in* $\mathcal{A}^\Delta_{\text{rob}}$ *without reaching an unsafe state. For any delay* $d \in \mathbb{R}_{\geq 0}$*, all the input actions* $i?$ *firable from a state* $((q_i, r'_i), u_i + d)$ *are also firable in* $(q_i, \widetilde{u}_i + d)$ *since the guards are the same. Therefore,* $((q_i, r'_i), u_i + d) \xrightarrow{i?} ((q_{i+1}, r_{i+1}), u_{i+1})$*, and* $(q_i, \widetilde{u}_i + d) \xrightarrow{i?} (q_{i+1}, \widetilde{u}_{i+1})$*. Additionally,* $(q_{i+1}, \widetilde{u}_{i+1})$ *is also a winning state, since it is an outcome of the strategy* F*, and* $\widetilde{u}_{i+1} = u_{i+1}$*, since the same clocks are reset. This proves the induction hypothesis*
2. *Otherwise, output has a strategy that eventually performs an action* $o!$ *after a delay* a*. Let* $\widehat{\rho}_{i+1} = \widehat{\rho}_i \xrightarrow{a} ((q_i, r'_i), u_i + a) \xrightarrow{o!} ((q_{i+1}, r_{i+1}), u_{i+1})$ *(concealing* τ *transitions). This implies that there exists an edge* $e \in \mathcal{A}$ *that fires* $o!$ *in* q_{i+1}*, the following edges exists in* $\mathcal{A}^\Delta_{\text{rob}}$*:* $(q_i, \tau_{o!}, \varphi, \lambda \cup \{y\}, q^\alpha_{i_e})$*,* $(q_i, \tau_{o!}, \{y = \Delta\}, \{y\}, q^\beta_{i_e})$*,* $(q^\alpha_{i_e}, o!, \{y \leq \Delta\}, \{y\}, q_{i+1})$ *and* $(q^\beta_{i_e}, o!, \{y \leq \Delta\}, \{y\}, q_{i+1})$*. Since* $o!$ *is firable in* $((q_i, r'_i), \widehat{u}_i + a)$*, this means that* $F_{\text{rob}}((q_i, r'_i)) = o!$ *and* $u_i + a \models \lceil r'_i \rceil_\Delta$*. Then by construction of* F_{rob}*,* $F(q^\alpha_{i_e}, \widetilde{r}'_i) = \tau_{o!}$*, and there exists* $b \in \mathbb{R}_{\geq 0}$ *s.t.* $(q_i, \widetilde{u}_i) \xrightarrow{b} (q_i, \widetilde{u}_i + b)$ *(concealing* $\tau_{o!}$ *transitions), and* $\widetilde{u}_i + b \in \widetilde{r}'_i$*. Additionally,* $\forall \delta \in [\text{-}\Delta, \Delta]. (q_i, \widetilde{u}_i) \xrightarrow{b+\delta} (q_i, \widetilde{u}_i + b + \delta) \xrightarrow{o!}$*. This is in particular the case for* $b + \delta = a$ *since* $a \in \lceil r'_i \rceil_\Delta$*. Therefore* $(q_i, \widetilde{u}_i) \xrightarrow{a} \xrightarrow{o!} (q_{i+1}, \widetilde{u}_{i+1})$*, and* $(q_{i+1}, \widetilde{u}_{i+1})$ *is winning since it is an outcome of* F*, and* $\widetilde{u}_{i+1} = u_{i+1}$*, since the same clocks are reset. To finish the induction step, the same argument as in the first case is used to demonstrate that any input action firable from* $((q_i, r_i), u_i)$ *after some delay* d *is also firable from* $(q_i, \widetilde{u}_i)$*.* □

This construction shall serve as a tool for deciding robust consistency, synthesizing a robust implementation, and other operations of the specification theory with robustness which are detailed in next section.

Remark on completeness. The robust game automaton is not a complete technique to solve robust timed games. Indeed, our notion of robustness introduces perturbations on the syntax of the automaton, whereas the robust game automaton modifies the semantics of the game. Therefore there exists specifications



(a) Specification (b) Implementation

Figure 6: Specification with 1−robust implementation, but with no robust strategy in the robust game automaton.

that can be robustly implemented, although they will be judged as non robust using the robust game automaton construction. For instance, the specification in Fig. 6a can be robustly implemented with the implementation in Fig. 6b. But for $\Delta = 1$ no robust strategy exists in the robust game automaton since clock x is not reset.

5. Robust Consistency and Robust Compatibility

5.1. Robust Consistency

We now provide a method to decide the Δ-robust consistency of a specification and synthesize robust implementations by solving a robust timed safety game, in which the output player must avoid a set of immediate error states. From there, the computation of a robust strategy described in previous section provides a method to synthesize an implementation of the specification that is robust with respect to outputs enlargement.

To account for input restrictions, we increase the set of error states $\mathsf{err}^{\mathcal{S}}$. Intuitively a specification is Δ-robust with respect to input $i?$, if between enabling of any two $i?$ edges at least 2Δ time passes, during which the reaction to $i?$ is unspecified. So, if the two transitions triggers Δ-too-late and Δ-too-early (respectively), there is no risk that the reaction is resolved non-deterministically in the specification.

In our input-enabled setup, lack of reaction is modeled using transitions to the universal (unpredictable) state. Formally, we say that Δ-robust specifications should admit Δ-*latency* of inputs. A state (q, u) verifies the Δ-latency condition for inputs, iff for each edge $e = (q, i?, \varphi, c, q')$, where $q' \neq l_{\mathrm{u}}$ and e is enabled in (q, u) we have:

$$\forall d \in [0, 2\Delta]. \forall e' = (q, i?, \varphi, c, q'')$$
$$\text{if } e' \neq e \text{ and } (q, u) \xrightarrow{d} (q, u + d) \text{ and } e' \text{ is enabled in } (q, u + d) \text{ then } q'' = l_{\mathrm{u}}$$

Definition 15 *For a specification $\mathcal{S}$ and $\Delta \in \mathbb{R}_{\geq 0}$, the set $\mathsf{err}^{\mathcal{S}}_{\Delta}$ of error states for Δ-robust consistency is such that $(q, u) \in \mathsf{err}^{\mathcal{S}}_{\Delta}$ if one the following conditions is verified:*

- *Violates independent progress:*

$$(\exists d \in \mathbb{R}_{\geq 0}.\, (q,u) \overset{d}{\not\to}) \text{ and } (\forall d.\forall o!.\, (q,u) \overset{d}{\to} (q,u+d) \Rightarrow (q,u+d) \overset{o!}{\not\to})$$

- *Violates Δ-latency of inputs:* $\exists e = (q, i?, \varphi, c, q')$, *with* $q' \neq l_{\mathrm{u}}$, *enabled in* (q,u), *such that* $\exists d \in [0, 2\Delta].(q,u) \overset{d}{\to} (q,u+d)$ *and* $\exists e' = (q, i?, \varphi, c, q'')$ *enabled in* $(q, u+d)$, *with* $e' \neq e$ *and* $q'' \neq l_{\mathrm{u}}$.

Observe that $\mathsf{err}^{\mathcal{S}} \subseteq \mathsf{err}^{\mathcal{S}}_{\Delta}$, because the error condition with robustness is weaker than in the classical case (cf. page 12).

The Δ-robust consistency game $(\mathcal{S}, W^{\mathrm{o}}(\mathsf{err}^{\mathcal{S}}_{\Delta}))$ can be solved using the construction of Definition 14. This synthesizes a robust strategy F and its syntactic outcome $\mathcal{S}_F$. Then we build the robust implementation $\mathcal{I}_F$ by applying the following transformation to $\mathcal{S}_F$:

- When we apply a Δ-perturbation on $\mathcal{I}_F$, a state $((q,r),u)$ can be reached even if $u \notin r \vee r\nearrow$. However due to the region partitioning, the inputs edges available in $\mathcal{I}_F$ might not be firable from $r \vee r\nearrow$. Then, in order to check the robust satisfaction relation between $(\mathcal{I}_F)_\Delta$ and $\mathcal{S}$, we add additional input edges to $\mathcal{I}_F$: for each location (q,r) in $\mathcal{I}_F$, for each edge $e = ((q,r), i?, \varphi, \lambda, (q^*, r^*))$ (with $q^* \neq l_{\mathrm{u}}$), and for each location (q',r') linked to (q,r) by a sequence of τ transitions, we add an edge $e' = ((q',r'), i?, \varphi, \lambda, (q^*, r^*))$.

- To support restriction of input guards in $(\mathcal{I}_F)_\Delta$, we replaced in $\mathcal{I}_F$ all guards φ of edges $e = ((q,r), i?, \varphi, \lambda, (q',r'))$ with $q' \neq l_{\mathrm{u}}$ by their enlargement $\lceil \varphi \rceil_\Delta$. Guards on edges to the l_{u} location are adjusted in order to maintain action determinism and input-enableness.

Note that this construction adds many input edges to the implementation, out of which many are never enabled. This simplifies the construction and the proof of correctness. In practice, to efficiently synthesize implementations, coarser abstractions like zones should be used that do not include τ transitions, thus avoiding the multiplication of input edges.

Theorem 2 *Let $\mathcal{S}$ be a specification. If F is a robust winning strategy in the Δ-robust consistency game, then $\mathcal{I}_F$ $\mathbf{sat}_\Delta$ $\mathcal{S}$ and $\mathcal{S}$ is Δ-robust consistent.*

Proof 3 (Theorem 2) *$\mathcal{S}$ is Δ-robust consistent if it admits a Δ-robust implementation. $\mathcal{I}_F$ verifies the independent progress condition since it corresponds to the outcome of the strategy F that avoids the inconsistent states in $\mathcal{S}$. For the same reason it also verifies the Δ-latency condition, which permits to increase guards on input edges without adding non-determinism. Since F is a symbolic strategy it may authorize small delays in the regions where an output action must be fired, and therefore $\mathcal{I}_F$ may not be output urgent. However, any point in these regions can be freely chosen to concretely implement $\mathcal{I}_F$*

We check now that $\mathcal{I}_F$ $\mathbf{sat}_\Delta$ $\mathcal{S}$ *with the following relation*

$$R = \{(((q,r),u),(q,u)) \in [\![(\mathcal{I}_F)_\Delta]\!]_{sem} \times [\![\mathcal{S}]\!]_{sem}\}$$

Note that since F *is a robust winning strategy, the runs of* $\lceil \mathcal{S}_F \rceil^o_\Delta$ *(and by construction the ones of* $(\mathcal{I}_F)_\Delta$*) also belongs to* $\mathcal{S}$*. Finally we assume that* $\mathcal{S}$ *can accept* τ *transitions in any states as output transitions. Let consider* $(((q,r),u),(q,u)) \in R$*:*

1. *If* $((q,r),u) \xrightarrow{d}{}^{(I_F)_\Delta} ((q,r),u+d)$ *for some* $d \in \mathbb{R}_{\geq 0}$*, then since the runs of* $(\mathcal{I}_F)_\Delta$ *are included into the ones of* $\mathcal{S}$*, it is also the case that* $(q,u) \xrightarrow{d}{}^{S} (q,u+d)$*.*
2. *If* $((q,r),u) \xrightarrow{o!}{}^{(I_F)_\Delta} ((q',r'),u')$ *for some* $o! \in Act_o$*, then there exists and edge* $((q,r),o!,\varphi,\lambda,(q',r'))$ *in* $\mathcal{I}_F$ *and also in* $\mathcal{S}_F$*. It also means that there exists a similar edge* $(q,o!,\varphi,\lambda,q')$ *in* $\mathcal{S}$*. And since the runs of* $(\mathcal{I}_F)_\Delta$ *are included into the ones of* $\mathcal{S}$*, it implies that* $(q,u) \xrightarrow{o!}{}^{S} (q',u')$*.*
3. *If* $((q,r),u) \xrightarrow{\tau}{}^{(I_F)_\Delta} ((q,r'),u)$ *then by assumption* $(q,u) \xrightarrow{\tau}{}^{S} (q,u)$*.*
4. *If* $(q,u) \xrightarrow{i?}{}^{S} (q',u')$ *for some* $i? \in Act_i$*, then there exists and edge* $e = (q,i?,\varphi,\lambda,q')$ *such that* $u \models \varphi$*. There also exists an edge* $(q,[u]) \xrightarrow{i?}{}^{G} (q',[u'])$ *in the region graph of* $\mathcal{S}$*. If* $u \in r$ *then this edge also exists in* $\mathcal{I}_F$*. Otherwise* $u \in \lceil r \vee r\nearrow \rceil_\Delta$*. This means that there exists a sequence of* τ *transitions in* $\mathcal{I}_F$ *between* r *and* $[u]$*, and by construction the input edge is copied in each location along this sequence, including* (q,r)*.* $u \models \varphi$ *implies that* $u \models \lfloor \lceil \varphi \rceil_\Delta \rfloor_\Delta$*, which proves that it is firable. Therefore* $((q,r),u) \xrightarrow{i?}{}^{I_\Delta} ((q',[u']),u')$*.* □

5.2. Conjunction

A conjunction of two specifications captures the intersection of their implementation sets. The following conjunction operator has been proposed in [8]:

Definition 16 *Let* $\mathcal{S} = (Loc^S, q_0^S, Clk^S, E^S, Act, Inv^S)$ *and* $\mathcal{T} = (Loc^T, q_0^T, Clk^T, E^T, Act, Inv^T)$ *be specifications that share the same alphabet of actions* Act*.*
We define their conjunction, denoted $\mathcal{S} \wedge \mathcal{T}$*, as the TIOA* $(Loc, q_0, Clk, E, Act, Inv)$ *where* $Loc = Loc^S \times Loc^T$*,* $q_0 = (q_0^S, q_0^T)$*,* $Clk = Clk^S \uplus Clk^T$*,* $Inv((q_s,q_t)) = Inv(q_s) \wedge Inv(q_t)$*, and the set of edges is defined by the following rule:*

$$\text{if } (q_s,a,\varphi_s,\lambda_s,q_s') \in E^S \text{ and } (q_t,a,\varphi_t,\lambda_t,q_t') \in E^T \text{ then } ((q_s,q_t),a,\varphi_s \wedge \varphi_t, \lambda_s \cup \lambda_t, (q_s',q_t')) \in E$$

It turns out that this operator is robust, in the sense of precisely characterizing also the intersection of the sets of *robust* implementations. So not only conjunction is the greatest lower bound with respect to implementation semantics, but also with respect to the robust implementation semantics. More precisely:

Theorem 3 *For specifications* $\mathcal{S}$*,* $\mathcal{T}$ *and* $\Delta \in \mathbb{Q}_{>0}$*:*

$$[\![\mathcal{S} \wedge \mathcal{T}]\!]^\Delta_{\text{mod}} = [\![\mathcal{S}]\!]^\Delta_{\text{mod}} \cap [\![\mathcal{T}]\!]^\Delta_{\text{mod}}$$

The theorem is a direct extension for robust implementations of Theorem 6 in [8]. We remark that due to the monotonicity of the refinement (Property 1), we can use two different delays Δ_1 and Δ_2, such that:

$$[\![\mathcal{S}]\!]^{\Delta_1}_{\text{mod}} \cap [\![\mathcal{T}]\!]^{\Delta_2}_{\text{mod}} \supseteq [\![\mathcal{S} \wedge \mathcal{T}]\!]^{\max(\Delta_1,\Delta_2)}_{\text{mod}}$$

So requirements with different precision can be conjoined, by considering the smaller jitter. Robustness of the operator in Def. 16 is very fortunate. Thanks to this, large parts of implementation of theory of [8] can be reused.

5.3. Parallel Composition and Robust Compatibility

Composition is used to build systems from smaller units. Two specifications $\mathcal{S}$, $\mathcal{T}$ can be composed only iff $Act^{\mathcal{S}}_{\text{o}} \cap Act^{\mathcal{T}}_{\text{o}} = \emptyset$. Parallel composition is obtained in [8] by a product, where the inputs of one specification synchronize with the outputs of the other:

Definition 17 *Let $\mathcal{S} = (Loc^S, q_0^S, Clk^S, E^S, Act^S, Inv^S)$ and $\mathcal{T} = (Loc^T, q_0^T, Clk^T, E^T, Act^T, Inv^T)$ be two composable specifications.*
We define their parallel product, denoted $\mathcal{S} \parallel \mathcal{T}$, as the TIOA $(Loc, q_0, Clk, E, Act, Inv)$, where $Loc = Loc^S \times Loc^T$, $q_0 = (q_0^S, q_0^T)$, $Clk = Clk^S \uplus Clk^T$, $Act = Act_{\text{o}} \cup Act_{\text{i}}$ with $Act_{\text{o}} = Act^{\mathcal{S}}_{\text{o}} \uplus Act^{\mathcal{T}}_{\text{o}}$ and $Act_{\text{i}} = (Act^{\mathcal{S}}_{\text{i}} \setminus Act^{\mathcal{T}}_{\text{o}}) \cup (Act^{\mathcal{T}}_{\text{i}} \setminus Act^{\mathcal{S}}_{\text{o}})$, $Inv(q_s, q_t) = Inv(q_s) \wedge Inv(q_t)$, and the set of edges is defined by the three following rules:

- *If $(q_s, a, \varphi_s, \lambda_s, q'_s) \in E^S$ with $a \in Act^S \setminus Act^T$ then each $q_t \in Loc^T$ gives rise to an edge $((q_s, q_t), a, \varphi_s, \lambda_s, (q'_s, q_t)) \in E$;*
- *If $(q_t, a, \varphi_t, \lambda_t, q'_t) \in E^T$ with $a \in Act^T \setminus Act^S$ then each $q_s \in Loc^S$ gives rise to an edge $((q_s, q_t), a, \varphi_t, \lambda_t, (q_s, q'_t)) \in E$;*
- *If $(q_s, a, \varphi_s, \lambda_s, q'_s) \in E^S$ and $(q_t, a, \varphi_t, \lambda_t, q'_t) \in E^T$ with $a \in Act^S \cap Act^T$ then this gives rise to an edge $((q_s, q_t), a, \varphi_s \wedge \varphi_t, \lambda_s \cup \lambda_t, (q''_s, q'_t)) \in E$.*

Example 5 *The two timed specifications in Fig. 1a and 1b can be composed together by synchronizing the outputs* cof *and* tea *of the Machine with the inputs of the Researcher. The resulting TIOA is a timed specification whose input is* coin *and outputs are* pub, cof *and* tea. *This composition scheme is illustrated in the diagram of Fig. 7.*

In the input-enableness setting we model incompatibility by introducing a predicate describing undesirable states denoted by the set und. For example, a communication failure can be modeled by redirecting an input edge to an undesirable location. In general any reachability objective, for example given by a temporal logic property, can serve as the set of undesirable behaviors und. It is important that such behaviors are avoided during the composition. For doing so, we propose to follow the optimistic approach to composition introduced in [9] that is *two specifications can be composed if there exists at least one environment*

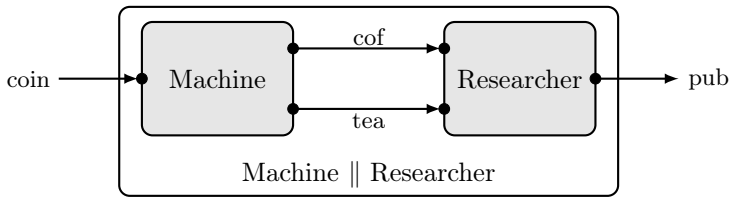


Figure 7: Diagram for the parallel composition of a coffee machine and a researcher

in which they can work together. In the robustness setting we consider imprecise environments by applying a Δ-perturbation to their outputs. Then, in what follows, we say that a specification is Δ-*robust useful* if there exists an imprecise environment $\mathcal{E}$ that avoids the undesirable states, whatever the specification does.

Definition 18 *A specification $\mathcal{S}$ is* Δ-robust useful *if there exists an environment $\mathcal{E}$ such that no undesirable states are reached in $[\![\lceil\mathcal{E}\rceil^o_\Delta \parallel \mathcal{S}]\!]_{\text{sem}}$.*

Property 4 (Monotonicity) *Given $\Delta_1 \leq \Delta_2$, if $\mathcal{S}$ be a Δ_2-robust useful specification, then $\mathcal{S}$ is Δ_1-robust useful.*

Proof 4 (Property 4) *If $\mathcal{S}$ be a Δ_2-robust useful this means that there exists an environment $\mathcal{E}$ such that no undesirable states is reached in $[\![\lceil\mathcal{E}\rceil^o_{\Delta_2} \parallel \mathcal{S}]\!]_{\text{sem}}$. Then for $\Delta_1 \leq \Delta_2$, $[\![\lceil\mathcal{E}\rceil^o_{\Delta_1}]\!]_{\text{sem}} \subseteq [\![\lceil\mathcal{E}\rceil^o_{\Delta_2}]\!]_{\text{sem}}$. This implies that $[\![\lceil\mathcal{E}\rceil^o_{\Delta_1} \parallel \mathcal{S}]\!]_{\text{sem}} \subseteq [\![\lceil\mathcal{E}\rceil^o_{\Delta_2} \parallel \mathcal{S}]\!]_{\text{sem}}$, which proves that $\mathcal{S}$ is also Δ_1-robust useful.* □

To check robust usefulness we solve the robust game $(\mathcal{S}, W^{\text{i}}(\mathsf{und}))$, and determine if the input player has a robust strategy F that avoids the undesirable states. Let $\mathcal{S}_F$ be the syntactic outcome of F in $\mathcal{S}$. We build from $\mathcal{S}_F$ a robust environment $\mathcal{E}_F$ by permuting the input and output players, such that each input in $\mathcal{S}_F$ becomes an output, and conversely.

Theorem 4 *Let $\mathcal{S}$ be a specification. If F is a robust winning strategy in the Δ-robust usefulness game, then $\mathcal{S}$ is Δ-robust useful in the environment $\mathcal{E}_F$.*

Proof 5 (Theorem 4) *The theorem directly follows from the definition of the robust strategy. $[\![\lceil\mathcal{S}_F\rceil^i_\Delta]\!]_{\text{sem}} \subseteq [\![\mathcal{S}]\!]_{\text{sem}}$ which implies that $\lceil\mathcal{E}_F\rceil^o_\Delta$ synchronizes with every action of $\mathcal{S}$ in their parallel product. Therefore, only the states belonging to $[\![\lceil\mathcal{S}_F\rceil^i_\Delta]\!]_{\text{sem}}$ can be reached in the composition, and by definition they are not undesirable.* □

Finally, two specifications are compatible if their composition is useful.

Definition 19 *Two composable specifications $\mathcal{S}$ and $\mathcal{T}$ are* Δ-robust compatible *if and only if $\mathcal{S} \parallel \mathcal{T}$ is Δ-robust useful.*

We now study the impact of adding Δ-perturbations introduced in previous section over parallel composition. It is important that the robust theory does not modify the definition of the operations themselves. This means that all the important properties of composition introduced in [8] remain valid. Moreover, robustness distributes over parallel composition in the following fashion:

Lemma 1 *For any implementations $\mathcal{I}$, $\mathcal{J}$ and a delay $\Delta \in \mathbb{Q}_{>0}$:*

$$(\mathcal{I} \parallel \mathcal{J})_\Delta \leq \mathcal{I}_\Delta \parallel \mathcal{J}_\Delta$$

Proof 6 (Lemma 1) *In the following, we denote $(Loc^k, q_0^k, Clk^k, E^k, Act^k, Inv^k)$, with $k \in \{I, J, I \parallel J\}$, the TIOAs corresponding to $\mathcal{I}, \mathcal{J}$, or $\mathcal{I} \parallel \mathcal{J}$, respectively, and $(St^k, (q_0^k, 0), Act^k, \rightarrow^k)$, with $k \in \{I, J, I \parallel J\}$, their semantics, and with $k \in \{I_\Delta, J_\Delta, [I \parallel J]_\Delta\}$, their perturbed semantics.*
First let recall Theorem 11 from [8] that states that $[\![\mathcal{I}_\Delta \parallel \mathcal{J}_\Delta]\!]_{\text{sem}} = [\![\mathcal{I}_\Delta]\!]_{\text{sem}} \mid [\![\mathcal{I}_\Delta]\!]_{\text{sem}}$, where $\mid$ is the parallel composition between TIOTS. Then we need to prove the refinement $[\![(\mathcal{I} \parallel \mathcal{J})_\Delta]\!]_{\text{sem}} \leq [\![\mathcal{I}_\Delta]\!]_{\text{sem}} \mid [\![\mathcal{J}_\Delta]\!]_{\text{sem}}$ by witnessing the following relation:

$$\begin{aligned} R = \{&(((q_i, q_j), u_{ij}), ((\widehat{q}_i, u_i), (\widehat{q}_j, u_j)) \in St^{[I\parallel J]_\Delta} \times (St^{I_\Delta} \times St^{J_\Delta}) \mid \\ &((q_i = \widehat{q}_i \wedge u_i = u_{ij|Clk^I}) \vee \widehat{q}_i = l_{\text{u}}) \wedge ((q_j = \widehat{q}_j \wedge u_j = u_{ij|Clk^J}) \vee \widehat{q}_j = l_{\text{u}})\} \end{aligned}$$

We prove by coinduction that R is a timed alternating relation. Let $(((q_i, q_j), u_{ij}), ((\widehat{q}_i, u_i), (\widehat{q}_j, u_j)) \in R$.

1. *If $((q_i, q_j), u_{ij}) \xrightarrow{d}{}^{[I\parallel J]_\Delta} ((q_i, q_j), u_{ij} + d)$ for some $d \in \mathbb{R}_{\geq 0}$, then by definition $u_{ij} \models \lceil Inv(q_i, q_j) \rceil_\Delta$. By construction of the parallel product $\lceil Inv(q_i, q_j) \rceil_\Delta = \lceil Inv(q_i) \wedge Inv(q_j) \rceil_\Delta = \lceil Inv(q_i) \rceil_\Delta \wedge \lceil Inv(q_j) \rceil_\Delta$. Then, we can deduce that $u_i + d \models \lceil Inv(q_i) \rceil_\Delta$ and $u_j + d \models \lceil Inv(q_j) \rceil_\Delta$. This implies that $(q_i, u_i) \xrightarrow{d}{}^{I_\Delta} (q_i, u_i + d)$ and $(q_j, u_j) \xrightarrow{d}{}^{J_\Delta} (q_j, u_j + d)$. Besides, by definition of the universal state, for any valuation u of a TIOA is always true that $(l_{\text{u}}, u) \xrightarrow{d} (l_{\text{u}}, u + d)$.*
   By definition of $[\![\mathcal{I}_\Delta]\!]_{\text{sem}} \mid [\![\mathcal{J}_\Delta]\!]_{\text{sem}}$, $((\widehat{q}_i, u_i), (\widehat{q}_j, u_j)) \xrightarrow{d}{}^{I_\Delta | J_\Delta} ((\widehat{q}_i, u_i + d), (\widehat{q}_j, u_j + d))$, and the relation R is trivially preserved in the next states.
2. *If $((q_i, q_j), u_{ij}) \xrightarrow{o!}{}^{[I|J]_\Delta} ((q_i', q_j'), u_{ij}')$, then $\exists e \in E^{I\parallel J}. e = ((q_i, q_j), o!, \varphi, c, (q_i', q_j'))$ such that $u_{ij} \models \lceil \varphi \rceil_\Delta$ and $u_{ij}' \models \lceil Inv(q_i', q_j') \rceil_\Delta$,*
   (a) *And if $o \in Act_o^I \backslash Act_i^J$ (or conversely, if $o \in Act_o^J \backslash Act_i^I$; this case is similar, so we will not consider it), then $\exists e_i \in E^I. e_i = (q_i, o!, \varphi_i, c_i, q_i')$, and $q_j' = q_j$, $\varphi = \varphi_i$, $c = c_i$, and $u'_{ij|Clk^J} = u_j$. Consequently, $u_i \models \lceil \varphi_i \rceil_\Delta$ and $u_i' = u'_{ij|Clk^I} \models \lceil Inv(q_i') \rceil_\Delta$, which proves that $(q_i, u_i) \xrightarrow{o!}{}^{I_\Delta} (q_i', u_i')$. Besides, by definition of the universal state, $(l_{\text{u}}, u_i) \xrightarrow{o!}{}^{I_\Delta} (l_{\text{u}}, u_i)$.*
   Then $((\widehat{q}_i, u_i), (\widehat{q}_j, u_j)) \xrightarrow{o!}{}^{I_\Delta | J_\Delta} ((\widehat{q}_i', u_i'), (\widehat{q}_j, u_j))$. Moreover, either $\widehat{q}_i = q_i$ and then $\widehat{q}_i' = q_i'$, or $\widehat{q}_i = l_{\text{u}} = \widehat{q}_i'$, which proves that the relation R is preserved.

(b) *If* $o \in Act_o^I \cap Act_i^J$ *(the reverse case* $o \in Act_o^J \cap Act_i^I$ *is similar), then* $\exists e_i \in E^I.\, e_i = (q_i, o?, \varphi_i, c_i, q_i')$ *and* $\exists e_j \in E^J.\, e_j = (q_j, o!, \varphi_j, c_j, q_j')$ *and* $\varphi = \varphi_i \wedge \varphi_j$, $c = c_i \cup c_j$. *On the side of* $\mathcal{I}$, *since* $u_{ij} \models \lceil \varphi \rceil_\Delta$ *and* $u'_{ij} \models \lceil Inv(q_i', q_j') \rceil_\Delta$, *we get that* $u_i \models \lceil \varphi_i \rceil_\Delta$ *and* $u_i' = u'_{ij|\,Clk^I} \models \lceil Inv(q_i') \rceil_\Delta$, *which implies that* $(q_i, u_i) \xrightarrow{o!}{}^{I_\Delta} (q_i', u_i')$.
On the side of $\mathcal{J}$, *we also get that* $u_j \models \lceil \varphi_j \rceil_\Delta$ *and* $u_j' \models \lceil Inv(q_j') \rceil_\Delta$. *If moreover* $u_j \models \lfloor \varphi_j \rfloor_\Delta$, *then as previously it implies that* $(q_j, u_j) \xrightarrow{o?}{}^{J_\Delta} (q_j', u_j')$. *Otherwise, a reductio ab absurdum argument allows us to prove that there exists no other edges* $e_j' = (q_j, o?, \varphi_j', c_j', q_j'') \in E^J$ *such that* $u_j \models \lfloor \varphi_j' \rfloor_\Delta$ *(since* $u_j \models \lceil \varphi_j \rceil_\Delta$ *it implies that* $\exists \varepsilon \in [\text{-}\Delta, \Delta].u_j + \varepsilon \models \varphi_j$, *but in the same time it would be the case that* $u_j + \varepsilon \models \varphi_j'$. *This is absurd since* J, *as an implementation is supposed to be deterministic). However, by construction of* $[\![J_\Delta]\!]_{\text{sem}}$ *, input-enableness is preserved by linking the unexpected input to a universal location* l_{u}. *So* $(q_j, u_j) \xrightarrow{o?}{}^{J_\Delta} (\mathsf{univ}, u_j)$.
Then $((\widehat{q}_i, u_i), (\widehat{q}_j, u_j)) \xrightarrow{o!}{}^{I_\Delta | J_\Delta} ((\widehat{q}_i', u_i'), (\widehat{q}_j', u_j'))$, *and as previously, by construction we check that the relation is preserved.*

3. *Finally, if* $((\widehat{q}_i, \widehat{u}_i), (\widehat{q}_j, \widehat{u}_j)) \xrightarrow{i?}{}^{I_\Delta | J_\Delta} ((\widehat{q}_i', u_i'), (\widehat{q}_j', u_j'))$, *for* $i \in Act_i^I \cap Act_i^J$ *(the cases* $i \in Act_i^I \backslash Act_i^J$ *or* $i \in Act_i^J \backslash Act_i^I$ *can be proved similarly by considering that only one component reacts, while the other stay in the same state), then from the definition of the composition:*

   - $(\widehat{q}_i, \widehat{u}_i) \xrightarrow{i?}{}^{I} (\widehat{q}_i', \widehat{u}_i')$, *and*
   - $(\widehat{q}_j, \widehat{u}_j) \xrightarrow{i?}{}^{J} (\widehat{q}_j', \widehat{u}_j')$.

   Besides, due to input-enableness, $((q_i, q_j), u_{ij}) \xrightarrow{i?}{}^{[I \| J]_\Delta} ((q_i, q_j), u_{ij} + d)$, *which implies that:*

   - $(q_i, u_i) \xrightarrow{i?}{}^{I} (q_i', u_i')$, *with* $u_i' = u'_{ij|\,Clk^I}$, *and*
   - $(q_j, u_j) \xrightarrow{i?}{}^{J} (q_j', u_j')$, *with* $u_j' = u'_{ij|\,Clk^J}$.

   If $\widehat{q}_i = l_{\text{u}}$ *then* $\widehat{q}_i' = l_{\text{u}}$, *and in this case the relation* R *is always preserved (and similarly for* $\widehat{q}_j$*). Otherwise* $\widehat{q}_i = q_i$ *and* $\widehat{u}_i = u_i$ *(and similarly for* $\widehat{q}_j$*). In this latter case, since* $[\![\mathcal{I}_\Delta]\!]_{sem}$ *is deterministic, it implies that* $\widehat{q}_i' = q_i'$ *and* $\widehat{u}_i' = u_i'$, *which also proves the induction for relation* R. □

Finally, we show in Theorem 5 that the independent implementability property of [8] can be extended to robust implementability, which follows from Lemma 1 and Theorem 10 in [8].

Theorem 5 *Let* $\mathcal{S}$ *and* $\mathcal{T}$ *be composable specifications and let* $\mathcal{I}$ *and* $\mathcal{J}$ *be* Δ*-robust implementations of* $\mathcal{S}$ *and* $\mathcal{T}$ *(resp.), i.e.* $\mathcal{I}$ $\mathbf{sat}_\Delta$ $\mathcal{S}$ *and* $\mathcal{J}$ $\mathbf{sat}_\Delta$ $\mathcal{S}$.
Then $\mathcal{I} \parallel \mathcal{J}$ $\mathbf{sat}_\Delta$ $\mathcal{S} \parallel \mathcal{T}$. *Moreover if* $\mathcal{S}$ *and* $\mathcal{T}$ *are* Δ*-compatible then* $\mathcal{I}$ *and* $\mathcal{J}$ *are also* Δ*-compatible.*

Proof 7 (Theorem 5) *The first part of the theorem is deduced from previous results:*

- *Due to lemma 1,* $[\![(\mathcal{I} \parallel \mathcal{J})_\Delta]\!]_{\text{sem}} \leq [\![\mathcal{I}_\Delta]\!]_{\text{sem}} \mid [\![\mathcal{J}_\Delta]\!]_{\text{sem}}$.
- *Then due Theorem 10 in [8],* $[\![\mathcal{I}_\Delta]\!]_{\text{sem}} \mid [\![\mathcal{J}_\Delta]\!]_{\text{sem}} \leq [\![\mathcal{S}]\!]_{\text{sem}} \mid [\![\mathcal{T}]\!]_{\text{sem}}$.
- *Finally due Theorem 11 in [8],* $[\![\mathcal{S}]\!]_{\text{sem}} \mid [\![\mathcal{T}]\!]_{\text{sem}} = [\![\mathcal{S} \parallel \mathcal{T}]\!]_{\text{sem}}$.

This proves $[\![(\mathcal{I} \parallel \mathcal{J})_\Delta]\!]_{\text{sem}} \leq [\![\mathcal{S} \parallel \mathcal{T}]\!]_{\text{sem}}$*, and therefore* $\mathcal{I} \parallel \mathcal{J}$ $\mathbf{sat}_\Delta$ $\mathcal{S} \parallel \mathcal{T}$.

We now prove the second part of the theorem. Since $\mathcal{S}$ *and* $\mathcal{T}$ *are* Δ*-compatible, there exists an environment* $\mathcal{E}$ *such that* $\lceil \mathcal{E} \rceil^o_\Delta \parallel (\mathcal{S} \parallel \mathcal{T})$ *avoids reaching any undesirable states. From Theorems 10 and 11 in [8], we get that* $\mathcal{I} \parallel \mathcal{J} \leq \mathcal{S} \parallel \mathcal{T}$*. Again with Theorem 10 we get that* $(\mathcal{I} \parallel \mathcal{J}) \parallel \lceil \mathcal{E} \rceil^o_\Delta \leq (\mathcal{S} \parallel \mathcal{T}) \parallel \lceil \mathcal{E} \rceil^o_\Delta$*. Consequently, since no undesirable states are reached in* $(\mathcal{S} \parallel \mathcal{T}) \parallel \lceil \mathcal{E} \rceil^o_\Delta$ *this is also the case in* $(\mathcal{I} \parallel \mathcal{J}) \parallel \lceil \mathcal{E} \rceil^o_\Delta$*, which proves that* $\mathcal{I} \parallel \mathcal{J}$ *is* Δ*-useful and so* $\mathcal{I}$ *and* $\mathcal{J}$ *are* Δ*-compatible.* □

Additionally, due to the monotonicity of perturbations with respect to the refinement, two different delays can be used to implement specifications $\mathcal{S}$ and $\mathcal{T}$. For two implementations $\mathcal{I}$ $\mathbf{sat}_{\Delta_1}$ $\mathcal{S}$ and $\mathcal{J}$ $\mathbf{sat}_{\Delta_2}$ $\mathcal{T}$ of the parallel components, their composition satisfies the composition of specifications with the smaller of the two precisions:

$$\mathcal{I} \parallel \mathcal{J} \ \mathbf{sat}_{\min(\Delta_1,\Delta_2)} \ \mathcal{S} \parallel \mathcal{T}$$

5.4. Quotient

Quotient is a dual operator to composition, such that for a large specification $\mathcal{T}$ and a small one $\mathcal{S}$, $\mathcal{T} \setminus\!\setminus \mathcal{S}$ is the specification of the components that composed with $\mathcal{S}$ will refine $\mathcal{T}$. In other words, $\mathcal{T} \setminus\!\setminus \mathcal{S}$ specifies the component that still needs to be implemented after having an implementation of $\mathcal{S}$, in order to build an implementation of $\mathcal{T}$. One possible application is when $\mathcal{T}$ is a system specification, and $\mathcal{S}$ is the plant, then a robust controller for a safety objective can be achieved by finding a Δ-consistent implementation of the quotient $\mathcal{T} \setminus\!\setminus \mathcal{S}$.

To apply quotienting, we require that $Act^S \subseteq Act^T$ and $Act^S_{\text{o}} \subseteq Act^T_{\text{o}}$. The construction of a quotient requires the use of a universal location l_{u}, as well as an inconsistent location $l_\emptyset$ that forbids any outputs and forbids elapsing of time.

Definition 20 *Let* $\mathcal{S} = (Loc^S, q_0^S, Clk^S, E^S, Act^S, Inv^S)$ *and* $\mathcal{T} = (Loc^T, q_0^T, Clk^T, E^T, Act^T, Inv^T)$ *be two specifications, with* $Act^S \subseteq Act^T$ *and* $Act^S_{\text{o}} \subseteq Act^T_{\text{o}}$*. Their* quotient*, denoted* $\mathcal{T} \setminus\!\setminus \mathcal{S}$*, is the TIOA* $(Loc, q_0, Clk, E, Act, Inv)$ *where* $Loc = Loc^T \times Loc^S \cup \{l_{\text{u}}, l_\emptyset\}$*,* $q_0 = (q_0^T, q_0^S)$*,* $Clk = Clk^T \uplus Clk^S \uplus \{x_{new}\}$*,* $Act = Act_{\text{i}} \uplus Act_{\text{o}}$ *with* $Act_{\text{i}} = Act^T_{\text{i}} \cup Act^S_{\text{o}} \cup \{i_{new}\}$ *and* $Act^T_{\text{o}} \setminus Act^S_{\text{o}}$*,* $Inv(q_t, q_s) = Inv(l_{\text{u}}) = true$ *and* $Inv(l_\emptyset) = \{x_{new} \leq 0\}$*, and the set* E *of edges is defined by the following rules:*

- $\forall q_t \in Loc^T . \forall q_s \in Loc^S . \forall a \in Act . \exists((q_t, q_s), a, \neg Inv^S(q_s), \{x_{new}\}, l_{\text{u}}) \in E$,
- $\forall q_t \in Loc^T . \forall q_s \in Loc^S . \exists((q_t, q_s), i_{new}, \neg Inv(q_t) \wedge Inv(q_s), \{x_{new}\}, l_\emptyset) \in E$,
- *If* $(q_t, a, \varphi_t, \lambda_t, q'_t) \in E^T$ *and* $(q_s, a, \varphi_s, \lambda_s, q'_s) \in E^S$*,*
  then $\exists((q_t, q_s), a, \varphi^T \wedge \varphi^S, \lambda_t \cup \lambda_s, (q'_t, q'_s)) \in E$,

- $\forall (q_s, a, \varphi_s, \lambda_s, q'_s) \in E^S$ *with* $a \in Act_{\mathrm{o}}^S$, $\exists((q_t, q_s), a, \varphi^S \wedge \neg G^T, \{x_{new}\}, l_\emptyset) \in E$, *where* $G^T = \bigvee\{\varphi_t \mid (q_t, a, \varphi_t, \lambda_t, q'_t)\}$,
- $\forall (q_t, a, \varphi_t, \lambda_t, q'_t) \in E^T$ *with* $a \notin Act^S$, $\exists((q_t, q_s), a, \varphi^T, \lambda_t, (q'_t, q'_s)) \in E$,
- $\forall (q_t, a, \varphi_t, \lambda_t, q'_t) \in E^T$ *with* $a \in Act_{\mathrm{o}}^S$, $\exists((q_t, q_s), a, \neg G^T, \{\}, l_{\mathrm{u}}) \in E$, *where* $G^T = \bigvee\{\varphi_s \mid (q_s, a, \varphi_s, \lambda_s, q'_s)\}$,
- $\forall a \in Act_{\mathrm{i}}. \exists(l_\emptyset, a, x_{new} = 0, \emptyset, l_\emptyset) \in E$,
- $\forall a \in Act. \exists(l_{\mathrm{u}}, a, true, \emptyset, l_{\mathrm{u}}) \in E$.

As stated in Theorem 12 of [8], the quotient gives a maximal (the weakest) specification for a missing component. This theorem can be generalized to specifications that are locally consistent (see [8]), and used to argue for completeness of the quotient construction in the robust case. It turns out that this very operator is also maximal for the specification of a robust missing component, in the following sense:

Theorem 6 *Let $\mathcal{S}$ and $\mathcal{T}$ be two specifications such that the quotient $\mathcal{T} \setminus\!\setminus \mathcal{S}$ is defined and let $\mathcal{J}$ be an implementation, then:*

$$\mathcal{S} \parallel \mathcal{J}_\Delta \leq \mathcal{T} \quad \textit{iff} \quad \mathcal{J} \ \mathbf{sat}_\Delta \ \mathcal{T} \setminus\!\setminus \mathcal{S}$$

Proof 8 (Theorem 6) *First let remark that $\mathcal{J}_\Delta$ is a locally consistent specification, as defined in [8]. Then, we can apply Theorem 12 of [8] to $\mathcal{J}_\Delta$ which proves:*

$$\mathcal{S} \parallel \mathcal{J}_\Delta \leq \mathcal{T} \quad \textit{iff} \quad \mathcal{J}_\Delta \leq \mathcal{T} \setminus\!\setminus \mathcal{S}$$

According to the definition of Δ-robust satisfaction this proves the theorem.

6. Counter Strategy Refinement For Parametric Robustness

In the previous sections we define and solve robustness problems for a fixed delay, and we study the properties of these perturbations with respect to the different operators in the specification theory. To complete this work we propose a technique that evaluates the greatest possible value of the perturbation. We follow a counterexample refinement approach, a technique used for abstraction refinement and called CEGAR in [28]. In our setting counterexamples are spoiling strategies computed for a given value of the perturbation. We replay these strategies on a parametric model of the robust game in order to refine the value of the perturbation.

The robustness problems that we consider in this sections are the parametric extension of the previously defined problems:

Robust Consistency. Given a specification $\mathcal{S}$, determine the greatest value of Δ such that $\mathcal{S}$ is Δ-robust consistent.

Robust Usefulness. Given a specification $\mathcal{S}$, determine the greatest value of Δ such that $\mathcal{S}$ is Δ-robust useful.

6.1. Parametric Timed Games

When we consider Δ as a free parameter, the robust game automaton construction of Section 4 defines a Parametric Timed I/O Automata, in a similar manner as Parametric Timed Automata are defined in [29, 30]. 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 21 *A* Parametric TIOA *with parameter Δ, is a TIOA $\mathcal{A}$ such that guards and invariants are replaced by parametric guards.*

For a given value $\delta \in \mathbb{Q}$ of the parameter, we define the non-parametric game $\mathcal{A}_\delta$ obtained by replacing each occurrence of the parameter Δ in the parametric guards of $\mathcal{A}$ 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 $[\![\mathcal{A}_\delta]\!]_{\text{sem}}$. 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:

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

6.2. Parametric Robustness Evaluation

Let Δ_{max} be the theoretical maximum value for Δ is the robust game (or its supremum). 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 [29]. In this paper we propose to compute an approximation of this maximum value. Due to the monotonicity of the robustness problems (Properties 1 and 4), 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 $(\mathcal{A}^{\Delta}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$ for output (again it applies symmetrically to input).

The algorithm assumes that the game $(\mathcal{A}^{0}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$ is won, and on contrary that $(\mathcal{A}^{\Delta_{init}}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$ is lost. At the heart of the algorithm the

Algorithm 1: Evaluation of the maximum robustness

Input: $(\mathcal{A}^{\Delta}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$: parametric robust timed game,
Δ_{init}: initial maximum value,
ϵ: precision
Output: Δ_{good} such that $\Delta_{max} - \Delta_{good} \leq \epsilon$

```
begin
    Δ_good ← 0
    Δ_bad ← Δ_init
    while Δ_bad − Δ_good > ε do
        (Δ_good, Δ_bad) ← RefineValues(A^Δ_rob, Δ_good, Δ_bad)
    end
    return Δ_good
end
```

procedure RefineValues plays the game for a chosen value, and update the variables Δ_{good} and Δ_{bad} according to the result.

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 $(\mathcal{A}^{\Delta_{mid}}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$. 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, which implies that they explore completely the state-graph of the model.

Correctness and termination. The algorithm is correct if two invariants are satisfied: Δ_{good} is a lower bound for Δ_{max}, and Δ_{bad} is a greater bound. These invariants must be preserved by the implementation of RefineValues.

The algorithm terminates if at each iteration RefineValues reduces the length of the interval $[\Delta_{good}, \Delta_{bad}]$ by some fixed minimum amount.

6.3. Counter Strategy Refinement

We propose an alternative method that analyzes the spoiling strategies computed when the game is lost and refine the value of the variable Δ_{bad}. With this algorithm only the last game is winning. The different steps are the following.

1. Play the game $(\mathcal{A}^{\Delta_{bad}}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$.
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 $\Delta_{bad} - \Delta_{min} > \epsilon$, return the values $(\Delta_{good}, \Delta_{min})$.
6. Else return the values $(\Delta_{good}, \Delta_{min} - \epsilon)$.

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 unfeasible. It takes as inputs the parametric game automaton $\mathcal{A}^{\Delta}_{\text{rob}}$, the symbolic graph $(\mathcal{Z}_{\mathcal{A}}^{\Delta_{bad}}, X_0, \rightarrow)$ computed for the game $(\mathcal{A}^{\Delta_{bad}}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$, 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 $(\mathcal{A}^{\Delta_{bad}}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$.

The algorithm is similar to the timed game algorithm proposed in [23] and implemented in the tool TIGA [31]. 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. First, for a symbolic state X we define its timed successors restricted by the symbolic strategy F_i (defined on a set of zones $\mathcal{Z}$) as:

$$\begin{aligned} X\nearrow^{F_i} = \{(q, u+d) \mid (q,u) \in X,\, d \in \mathbb{R}_{\geq 0}, \forall d' \in [0,d]. \\ \text{if } \exists Z \in \mathcal{Z}.\ \exists Z' \in \mathcal{Z}.\ u+d \in Z.\ u+d' \in Z' \\ \text{then } F((q,Z')) = F((q,Z)) \vee F((q,Z')) = \mathsf{delay}\} \end{aligned}$$

$\mathsf{Outcome}(F_i)$ 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.

Theorem 7 (Soundness) *Algorithm 2 returns a set of values $\Gamma \subseteq \mathbb{Q}$ such that $\forall \delta \in \Gamma$ the game $(\mathcal{A}^{\delta}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$ is lost.*

Proof 9 (Theorem 7) *For a value $\delta \in \mathbb{Q}$, Algorithm 2 is similar to the timed games algorithm of TIGA described in [23], but with less exploration steps since only the outcome of the spoiling strategy are explored. Therefore for that value $PWin[X_0] \subseteq Win[X_0]$.*

Ab absurdo, if $\delta \in \Gamma$ is a good value, i.e. $(\mathcal{A}^{\delta}_{\text{rob}}, W^{\text{o}}(\mathsf{Bad}))$ is winning, then $\mathbf{0} \notin Win[X_0]$, and consequently $\mathbf{0} \notin PWin[X_0]$. This implies that $\Gamma = \mathsf{Minimize}((PWin[X_0] \cap \mathbf{0})_{|\Delta}) = \emptyset$, which is a contradiction.

Theorem 7 ensures that RefineValues preserves the invariants of Algorithm 1.

Algorithm 2: Counter strategy refinement

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

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

begin
  `/* Initialisation */`
  $Waiting \leftarrow \emptyset$
  for $X = (q, Z) \in \mathcal{Z}_{\mathcal{A}}$ **do**
    if $q \in \mathsf{Bad}$ **then**
      $PWin[X] \leftarrow [\![Inv(q)]\!]$
      $Waiting \leftarrow Waiting \cup \{Y \mid \exists \rho.\, \rho \rightarrow Y \rightarrow X \in \mathsf{Outcome}(F_i)\}$
    else
      $PWin[X] \leftarrow \emptyset$
    end
  end
  `/* Backward exploration */`
  while $(Waiting \neq \emptyset) \wedge \mathbf{0} \notin PWin[X_0])$ **do**
    $X = (q, Z) \leftarrow \mathsf{pop}(Waiting)$
    $PBad^* \leftarrow \neg[\![Inv(q)]\!] \cup (\bigcup_X \xrightarrow{a \in Act_i}_Y \mathsf{PPred}_a(Win[Y]))$
    $PGood^* \leftarrow \bigcup_X \xrightarrow{a \in Act_o}_Y \mathsf{PPred}_a([\![Inv(Y)]\!] \setminus PWin[Y])$
    $PWin[X] \leftarrow \mathsf{PPred}_t(PBad^*, PGood^* \setminus PBad^*)$
    $Waiting \leftarrow Waiting \cup \{Y \mid \exists \rho.\, \rho \rightarrow Y \rightarrow X \in \mathsf{Outcome}(F_i)\}$
  end
  return $\mathsf{Minimize}((PWin[X_0] \cap \mathbf{0})_{|\Delta})$
end

7. Implementation and experiments

7.1. PyECDAR implementation

The specification theory described in [8] is implemented in the tool ECDAR [32]. 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 [33]. Inside this tool, the theory presented in Sections 4 and 5 is implemented as a set of model transformations:

1. Computation of $\mathcal{I}_{\Delta}$, the Δ-perturbation of an implementation $\mathcal{I}$ 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 [23] and the on-the-fly timed game algorithm presented in [25].

To solve the parametric robustness problems we have implemented the heuristic presented in Section 6 that approximates the maximum solution through a counter strategy refinement, and we have implemented a binary search heuristic to compare the efficiency. In the algorithm 2 operations on parametric symbolic states are handled with the Parma Polyhedra Library [34]. 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 [30].

7.2. 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_{init} and ϵ.

Specification of a university: . The toy examples featured in this paper are extracted from [8]. They are part of an overall specification of a university, composed by three specifications: the coffee machine (M) of Fig. 1a, the researcher (R) of Fig. 1b, and the Administration (A) (see [8]). 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_{init} 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_{init} 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.

Model	Game size loc.	trans.	$\Delta_{init} = 8$ $\epsilon = 0.1$ CR	BS	$\Delta_{init} = 6$ $\epsilon = 0.1$ CR	BS	$\Delta_{init} = 8$ $\epsilon = 0.01$ CR	BS	$\Delta_{init} = 6$ $\epsilon = 0.01$ 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 ‖ **A**	41	158	10.1s	10.1s	10.1s	9.6s	10.4s	17.5s	10.4s	17.6s
R ‖ **A**	48	201	14.1s	12.1s	12.5s	11s	14.1s	19.6s	12.5s	19.4s
M ‖ **R**	44	152	10s	15.5s	9.81s	15.8s	10.3s	22.9s	9.78s	29.2s
M ‖ **R** ‖ **A**	180	803	54.4s	56.3s	54.6s	112s	55s	58.8s	55.7s	216s

Table 1: Robust consistency of the university specifications

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

Table 2: Robust compatibility between the university specifications

Specification of a Milner Scheduler: . The second experiment studies a real time version of Milner's scheduler previously introduced in [32]. 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.

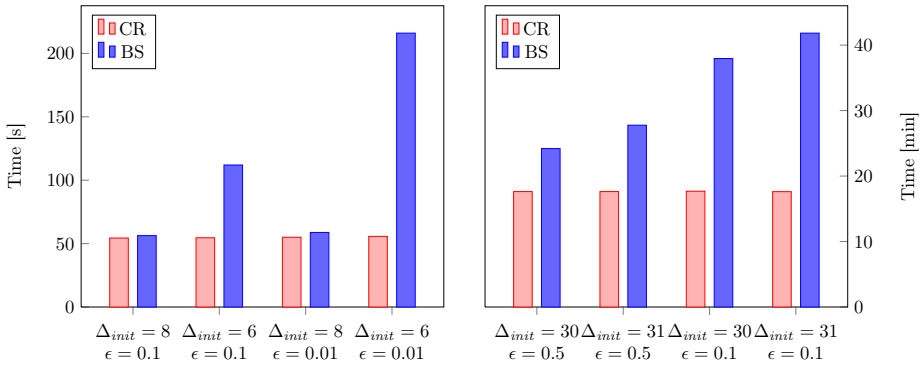
7.3. Interpretation

Previous results are sum up in Fig. 8 in order to compare the performances of the two methods on the most complex examples.

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

Model	Game size loc.	trans.	$\Delta_{init} = 30$ $\epsilon = 0.5$ CR	BS	$\Delta_{init} = 31$ $\epsilon = 0.5$ CR	BS	$\Delta_{init} = 30$ $\epsilon = 0.1$ CR	BS	$\Delta_{init} = 31$ $\epsilon = 0.1$ 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



(a) University experiment $M \parallel R \parallel A$ model

(b) Milner experiment 4 nodes model

Figure 8: Comparisons of the performances between the two methods Counter strategy Refinement (CR) and Binary Search (BS)

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_{init} 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 $\epsilon = 0.1$). 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_{init} 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_{init} 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 $\Delta_{min} - \epsilon$. The experiments shows this has no impact on the performances.

8. Concluding Remarks

We have presented a compositional framework for reasoning about robustness of timed I/O specifications. Our theory builds on the results presented in [8] combined together with a new robust timed game for robust specification theories. It can be used to synthesize an implementation that is robust with respect to a

given specification, and to combine or compare specifications in a robust manner. In our approach, robustness is achieved through syntactic transformations, which allows reusing classical analysis technique and tools. In particular we extend the construction of [11] to the setting of specification theories, to solve robust games by reducing them to problems on classical timed games.

As a new contribution from [22], we also study the parametric robustness problems and evaluate the maximum imprecision allowed by specifications. To this end we propose a counter example refinement approach that analyzes spoiling strategies in timed games. These contributions have been implemented in a prototype tool that has been used to evaluate the performances of our counter strategy refinement approach.

We have focused in this paper on solving robust consistency and robust compatibility problems. This provides a constructive approach to synthesize robust implementations. In a future version of our tool we would like to apply the counter example refinement approach on the alternating simulation game, in order to solve the parametric satisfaction problem for an existing implementation.

In future works we also plan to extend our approach to different models, like timed automata with stochastic semantics [35]. In this context we could give a stochastic definition of robustness that would allow a more expressive quantitative analysis than the worst case scenario used in this paper.

References

[1] O. M. Group, Corba 3.2, 2011. http://www.omg.org/spec/CORBA/3.2/.

[2] J. McAffer, P. VanderLei, S. Archer, OSGi and Equinox: Creating Highly Modular Java Systems, Addison-Wesley, Amsterdam, 2010.

[3] W3C, Web services description language (wsdl) version 2.0 part 1: Core language, 2007. http://www.w3.org/TR/wsdl20/.

[4] E. Standard, Functional safety of electrical/electronic/programmable electronic safety-related systems: IEC 61508-1:2010, DS/EN 61508, 2010.

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

[6] T. A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine, Symbolic model checking for real-time systems, Inf. Comput. 111 (1994) 193–244.

[7] G. Behrmann, A. David, K. G. Larsen, P. Pettersson, W. Yi, Developing uppaal over 15 years, Softw., Pract. Exper. 41 (2011) 133–142.

[8] A. David, K. G. Larsen, A. Legay, U. Nyman, A. Wąsowski, Timed I/O automata: a complete specification theory for real-time systems, in: HSCC, ACM, 2010, pp. 91–100.

[9] L. de Alfaro, T. A. Henzinger, Interface automata, in: ESEC / SIGSOFT FSE, pp. 109–120.

[10] L. de Alfaro, T. A. Henzinger, M. Stoelinga, Timed interfaces, in: EMSOFT, volume 2491 of *LNCS*, Springer, 2002, pp. 108–122.

[11] K. Chatterjee, T. A. Henzinger, V. S. Prabhu, Timed parity games: Complexity and robustness, in: FORMATS, volume 5215 of *LNCS*, Springer, Saint Malo, France, 2008, pp. 124–140.

[12] The COMBEST Consortium, Combest, 2008 – 2011. http://www.combest.eu.

[13] The SPEEDS Consortium, Speeds, 2006 – 2010. http://www.speeds.eu.com.

[14] E. Badouel, A. Benveniste, B. Caillaud, T. Henzinger, A. Legay, R. Passerone, Contract Theories for Embedded Systems : A white paper, Research report, IRISA/INRIA Rennes, 2009.

[15] A. Puri, Dynamical properties of timed automata, in: Formal Techniques in Real-Time and Fault-Tolerant Systems, volume 1486 of *LNCS*, Springer, 1998, pp. 210–227.

[16] M. D. Wulf, L. Doyen, J.-F. Raskin, Almost ASAP semantics: from timed models to timed implementations, Formal Aspects of Computing 17 (2005) 319–341.

[17] M. Wulf, L. Doyen, N. Markey, J.-F. Raskin, Robust safety of timed automata, Formal Methods in System Design 33 (2008) 45–84.

[18] P. Bouyer, K. G. Larsen, N. Markey, O. Sankur, C. Thrane, Timed automata can always be made implementable, in: CONCUR, volume 6901 of *LNCS*, Springer, Aachen, Germany, 2011, pp. 76–91.

[19] O. Sankur, P. Bouyer, N. Markey, Shrinking timed automata, in: FSTTCS, Leibniz International Proceedings in Informatics, Leibniz-Zentrum für Informatik, Mumbai, India, 2011, pp. 90–102.

[20] P. Bouyer, N. Markey, O. Sankur, Robust model-checking of timed automata via pumping in channel machines, in: FORMATS, volume 6919 of *LNCS*, Springer, Aalborg, Denmark, 2011, pp. 97–112.

[21] R. Jaubert, P.-A. Reynier, Quantitative robustness analysis of flat timed automata, in: FOSSACS, volume 6604 of *LNCS*, Springer, 2011, pp. 229–244.

[22] K. G. Larsen, A. Legay, L.-M. Traonouez, A. Wasowski, Robust specification of real time components, in: FORMATS, volume 6919 of *LNCS*, Springer, Aalborg, Denmark, 2011, pp. 129–144.

[23] F. Cassez, A. David, E. Fleury, K. G. Larsen, D. Lime, Efficient on-the-fly algorithms for the analysis of timed games, in: CONCUR, volume 3653 of *LNCS*, Springer, 2005, pp. 66–80.

[24] L. de Alfaro, T. A. Henzinger, Interface-based design, in: In Engineering Theories of Software Intensive Systems, Marktoberdorf Summer School.

[25] P. Bulychev, T. Chatain, A. David, K. G. Larsen, Efficient on-the-fly algorithm for checking alternating timed simulation, in: FORMATS, volume 5813 of *LNCS*, Springer, 2009, pp. 73–87.

[26] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, M. Stoelinga, The element of surprise in timed games, in: R. M. Amadio, D. Lugiez (Eds.), CONCUR, volume 2761 of *LNCS*, Springer, 2003, pp. 142–156.

[27] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems (an extended abstract), in: STACS, pp. 229–242.

[28] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement, in: CAV, volume 1855 of *LNCS*, Springer, 2000, pp. 154–169.

[29] R. Alur, T. A. Henzinger, M. Y. Vardi, Parametric real-time reasoning, in: STOC, pp. 592–601.

[30] T. Hune, J. Romijn, M. Stoelinga, F. W. Vaandrager, Linear parametric model checking of timed automata, J. Log. Algebr. Program. 52-53 (2002) 183–220.

[31] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, D. Lime, Uppaal-tiga: Time for playing games!, in: CAV, volume 4590 of *LNCS*, Springer, 2007, pp. 121–125.

[32] A. David, K. G. Larsen, A. Legay, U. Nyman, A. Wąsowski, ECDAR: An environment for compositional design and analysis of real time systems, in: ATVA, volume 6252 of *LNCS*, Springer, Singapore, 2010, pp. 365–370.

[33] Python implementation of ECDAR, Pyecdar, 2011. https://launchpad.net/pyecdar.

[34] R. Bagnara, P. M. Hill, E. Zaffanella, 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 (2008) 3–21.

[35] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. van Vliet, Z. Wang, Statistical model checking for networks of priced timed automata, in: FORMATS, volume 6919 of *LNCS*, Springer, Aalborg, Denmark, 2011, pp. 80–96.