Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Robust synthesis for real-time systems ☆

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

A B S T R A C T

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 for computing safe perturbation values.

© 2013 Elsevier B.V. All rights reserved.

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 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
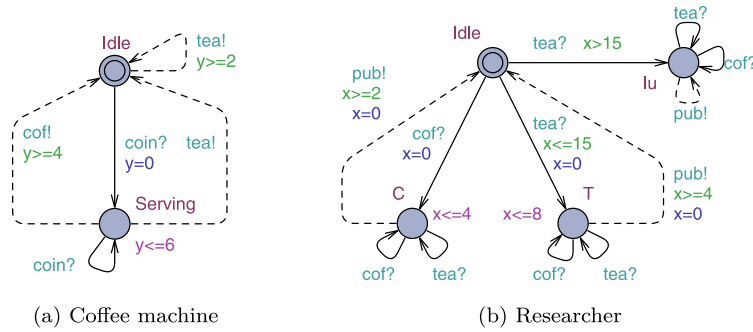
(a) Coffee machine　　　　　　　　(b) Researcher

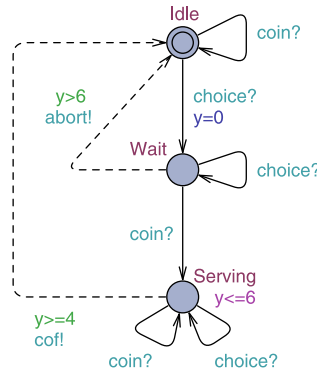**Fig. 1.** Timed specifications with timed I/O automata.

Fig. 2. Non-robust specification of a coffee machine.

[6]. Mature modeling and model-checking tools exist, such as Uppaal [7], that implement this technique and have been applied to numerous industrial applications [8–12].

In [13], 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. Fig. 1(a) displays a specification of a coffee machine that receives an input coin? and outputs either coffee (cof!) or tea (tea!). It can be composed with the specification of a researcher in Fig. 1(b) by synchronizing input with output labeled with the same channel name (cof and tea). In Fig. 1(b) 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 [14]. 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 [13] 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, delivers the coffee. If the payment does not arrive within six 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. Suppose that the environment makes the choice? action in the Idle location, waits six time units, and then provides the coin? action. In such execution the system reaches the location Serving with clock y at value 6. The invariant $y \leqslant 6$ in the Serving location requires now that any valid implementation must deliver the coffee (cof!) immediately, in zero time units. No physical system would permit this, so we say that this state is not robustly consistent.

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 behavior of the specification that the serving should take six 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 four time units. This would guarantee at least two time units for brewing the coffee. Despite both corrections being quite simple, it is clear that subtle timing mistakes like this one are very difficult 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 behaviors that themselves are robust. Examples exist where independent components appear compatible, but their compatibility requires infinitely precise execution platform or an infinitely fast environment.

The timing precision errors in specifications are not handled in any way in idealized interface theories such as [13,15]. These and similar issues have led 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:

- 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 relation 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 modifications to the definitions of the operators themselves and that all the good properties of a specification theory (including independent implementability) are maintained—the only effort requires is reproving the properties of the operators in a non-idealized setting.
- 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 [16], 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 [16] 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 computes the greatest admissible perturbations for the robustness problems. We apply a counterexample abstraction refinement-like technique that analyzes parametrically the results of loosing timed games in order to refine the value of the perturbation. This technique and the different constructions presented in the paper are implemented in a prototype tool. We demonstrate the performance of the refinement heuristic against the baseline of 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 [13], 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 [17–19].

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 Section 5 to reason about consistency, conjunction, parallel composition, in order to synthesize robust implementations of real-time components. In Section 6 we develop a counterexample refinement technique to measure the maximum 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 three 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 [20]. The problem has been linked to the implementation problem in [21], 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 [22] to propose another robust semantics with both clock drifts and guard enlargement. Extending [20] 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 [21,22] 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. [16], 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 [16] by allowing negative perturbation of delays.

Our paper is also similar to the works in [23,24] 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 [22] as the existence of a positive value for the imprecision of a timed automaton. 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 [25]. Finally a quantitative analysis is performed in [26] 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 computes 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 [27]. 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 [27] 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 [13]. We use $\mathbb{N}$ for the set of all natural numbers, $\mathbb{R}$ for the set of all real numbers, and $\mathbb{R}_{\geqslant 0}$ (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 $x \in \mathbb{R}_{\geqslant 0}$, 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 [13], 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 behaviors 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 $\rightarrow^S : St^S \times (\Sigma^S \cup \mathbb{R}_{\geqslant 0}) \times St^S$ 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 $\mathbb{R}_{\geqslant 0}$, 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 $d_1, d_2 \in \mathbb{R}_{\geqslant 0}$ 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 $s_1 \xrightarrow{a_1}^S s_2 \xrightarrow{a_2}^S \ldots s_{n-1} \xrightarrow{a_{n-1}}^S s_n \xrightarrow{a_n} \ldots$ such that for all $1 \leqslant i \leqslant n$, $s_i \xrightarrow{a_i}^S s_{i+1}$ with $a_i \in \Sigma^S \cup \mathbb{R}_{\geqslant 0}$. We write Runs$(s_1, S)$ for the set of runs of S starting in s_1 and Runs(S) for Runs(s_0, S). We write States(ρ) for the set of states reached in ρ, and if ρ is finite then we denote the last state occurring in ρ by last(ρ).

A TIOTS S is *deterministic* iff the action or delay fully determines the next state: $\forall a \in \Sigma^S \cup \mathbb{R}_{\geqslant 0}$, 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:

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

Output urgency captures predictability of timing of system's reactions. Finally, we say that a TIOTS S satisfies the *independent progress* condition iff it can always evolve using delays and outputs, regardless whether the environment collaborates

providing inputs or not. This is not a limiting assumption—in real systems, when the environment does not interact, the time is simply passing, and so it should be possible in the models of systems. Formally, for each state s we have either $(\forall d \geqslant 0.\exists s' s \xrightarrow{d}{}^S s')$ or $(\exists d \in \mathbb{R}_{\geqslant 0}.\exists o! \in \Sigma_o^S.\exists s', s'' \in St^S. \xrightarrow{d}{}^S s'$ and $s' \xrightarrow{o!}{}^S s'')$. The property guarantees 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 $Clk \mapsto \mathbb{R}_{\geqslant 0}$ (thus an element of $\mathbb{R}_{\geqslant 0}^{Clk}$). Given a valuation u and $d \in \mathbb{R}_{\geqslant 0}$, 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 $\mathcal{B}(Clk)$ 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 $\prec \in \{<, \leqslant, >, \geqslant\}$. By $\mathcal{U}(Clk) \subset \mathcal{B}(Clk)$, we denote the set of constraints restricted to upper bounds and without clock differences. For constraint $\varphi \in \mathcal{B}(Clk)$ and $u \in \mathbb{R}_{\geqslant 0}^{Clk}$, we write $u \models \varphi$ if u satisfies φ. If $Z \subseteq \mathbb{R}_{\geqslant 0}^{Clk}$, we write $Z \models \varphi$ if $u \models \varphi$ for all $u \in Z$. We write $[\![\varphi]\!]$ to denote the set of valuations $\{u \in \mathbb{R}_{\geqslant 0}^{Clk} \mid u \models \varphi\}$. A subset $Z \subseteq \mathbb{R}_{\geqslant 0}^{Clk}$ is a *zone* if $Z = [\![\varphi]\!]$ for some $\varphi \in \mathcal{B}(Clk)$. Let $Clk' \subset Clk$ and $Z \subseteq \mathbb{R}_{\geqslant 0}^{Clk}$ be a zone. We define the projection of Z on the subset of clocks Clk' as $Z_{|Clk'} = \{u' \in \mathbb{R}_{\geqslant 0}^{Clk'} \mid \exists u \in Z.u' = u_{|Clk'}\}$.

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*,
- $E \subseteq Loc \times Act \times \mathcal{B}(Clk) \times 2^{Clk} \times Loc$ is a set of *edges*,
- $Act = Act_i \uplus Act_o$ is a finite set of *actions*, partitioned into *inputs* (Act_i) and *outputs* (Act_o),
- $Inv : Loc \mapsto \mathcal{U}(Clk)$ is a set of location *invariants*.

Without loss of generality we assume that the guards are satisfiable and that the invariants are always satisfied by the incoming edges. Formally, let $e = (q, a, \varphi, \lambda, q') \in E$, we assume that $[\![\varphi]\!] \neq \emptyset$ and that $\forall u \in [\![\varphi]\!].u[\lambda] \models Inv(q')$.

A *universal location*, denoted l_u, in a TIOA accepts every input and can produce every output at any time. Formally l_u is such that, $\forall a \in Act.\exists (l_u, a, \top, \emptyset, l_u) \in E$, where $\top$ is the clock constraints such that $[\![\top]\!] = \mathbb{R}_{\geqslant 0}^{Clk}$. We assume that every TIOA contains a universal location, even if it is not drawn on the graph. The universal location will be used to model an unpredictable behavior of a component.

Example 1. Fig. 1(b) on page 97 depicts an example of a TIOA that admits two input actions cof? and tea?, and one output action pub!. Edges are labeled with an action, a guard and a set of reset clocks. Edges with input action are drawn with plain arrows, while Edges with output action are drawn with dashed arrows. Location are labeled with a name and an invariant constraint. lu is the universal location.

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

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

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

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

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

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

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

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

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:

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

These operations can be implemented symbolically on zones using Difference Bound Matrices (DBMs) [28].

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 $Z \subseteq \mathbb{R}_{\geqslant 0}^{Clk}$. 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 the guards and invariants 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 $v(x) > M$,
- $\forall x, y \in Clk$, $\forall k \in [-M, M]$, $u(x) - u(y) \leqslant k$ iff $v(x) - v(y) \leqslant k$,
- $\forall x \in Clk$ if $u(x) \leqslant M$ 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. Formally, $r \nearrow$ is the regions such that $\forall u \in r. \exists d > 0.(u + d \in r \nearrow \ \wedge \ \forall d' < d. u + d' \in r \cup r \nearrow)$. 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 $\mathbb{R}_{\geqslant 0}^{Clk}$. 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 $Z' = ((Z \cap [\![\varphi]\!])[\lambda]) \nearrow \cap [\![Inv(q')]\!]$.

3.2. Basics of the timed specification theory

In [13], 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 satisfies the output urgency and the independent progress conditions.

Example 3. The TIOA in Fig. 1(b) 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 Fig. 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 produces pub.

In specification theories, a *refinement* relation plays a central role. It allows to compare specifications, and to relate implementations to specifications. In [13], as well as in [14,29,30], refinement is defined in the style of alternating (timed) simulation:

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

- If $\exists i? \in \Sigma_i. \exists s' \in St^S. s \xrightarrow{i?}{}^S s'$, then $\exists t' \in St^T. t \xrightarrow{i?}{}^T t'$ and $(t', s') \in R$.
- If $\exists o! \in \Sigma_o. \exists t' \in St^T. t \xrightarrow{o!}{}^T t'$, then $\exists s' \in St^S. s \xrightarrow{o!}{}^S s'$ and $(t', s') \in R$.
- If $\exists d \geqslant 0. \exists t' \in St^T. t \xrightarrow{d}{}^T t'$, then $\exists s' \in St^S. s \xrightarrow{d}{}^S s'$ and $(t', s') \in R$.

---

[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.
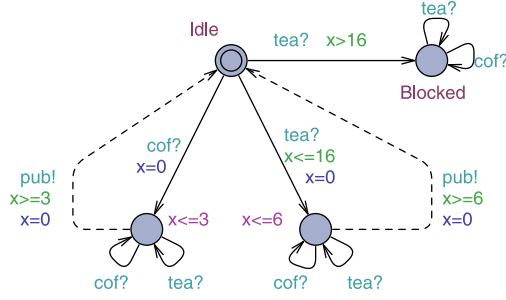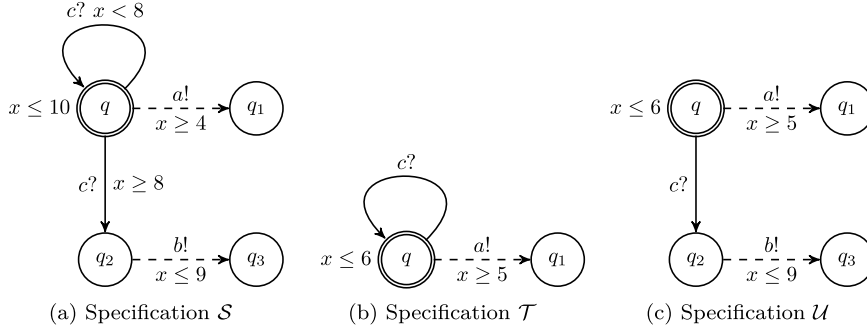
Fig. 3. Implementation for a researcher.

Fig. 4. Refinement between specifications: $\mathcal{T} \leqslant \mathcal{S}$, but $\mathcal{U} \not\leqslant \mathcal{S}$.

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

Example 4. We illustrate the concept of refinement between three simple specifications presented in Fig. 4. $\mathcal{T}$ refines $\mathcal{S}$, because it can only delay up to $x = 6$ and performs $a!$ between $[5, 6]$. However, $\mathcal{U}$ does not refine $\mathcal{S}$ (and $\mathcal{S}$ does not refine $\mathcal{U}$), because when $c?$ is received at $x = 0$, the states $(q, 0)$ from $\mathcal{S}$ and $(q_2, 0)$ from $\mathcal{U}$ must be in relation, which is not possible because $(q_2, 0) \xrightarrow{b!}^{\mathcal{U}} (q_3, 0)$ is not allowed at $(q, 0)$ by $\mathcal{S}$.

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

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

Definition 6. A specification $\mathcal{S}$ is *consistent* iff there exists an implementation $\mathcal{I}$ such that $\mathcal{I}$ **sat** $\mathcal{S}$.

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 Definition 5 that takes this into account. It is a fortunate property of Definition 4 that we do not need to modify it in order to reason about robust implementations (Property 3 in Section 4).

In [13], 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 [13], which will be illustrated in Section 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). The delay chosen by one player is implicitly defined by the time until a controllable action is chosen. 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
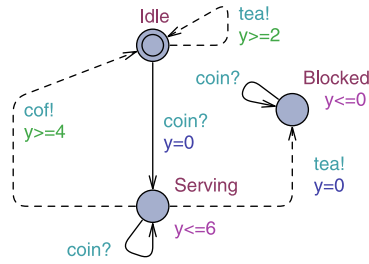
Fig. 5. Consistent specification with error states.

games [31]. 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 7. A memoryless *strategy* f_o for the output player on the TIOA $\mathcal{A}$ is a partial function $St^{[\![\mathcal{A}]\!]_{\text{sem}}} \mapsto Act_o \cup \{\text{delay}\}$, such that

- Whenever $f_o(s) \in Act_o$ then $s \xrightarrow{f_o(s)} s'$ for some s'.
- Whenever $f_o(s) = \text{delay}$ then $s \xrightarrow{d} s''$ for some $d > 0$ and state s'', and $f_o(s'') = \text{delay}$.

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

Definition 8. 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 $[\![\mathcal{A}]\!]_{\text{sem}}$. Outcome$(s, f_o, f_i)$ is the subset of Runs$(s, [\![\mathcal{A}]\!]_{\text{sem}})$ defined inductively by:

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

A *winning condition* for a player in the TIOA $\mathcal{A}$ is a subset of Runs$([\![\mathcal{A}]\!]_{\text{sem}})$. This player is then called the *verifier*, whereas the other player tries to prevent her from winning, and therefore is called the *spoiler*. In safety games the winning condition is to avoid a set Bad $\subseteq St^{[\![\mathcal{A}]\!]_{\text{sem}}}$ of "bad" states. Formally, the winning condition for output is $W^o(\text{Bad}) = \{\rho \in$ Runs$([\![\mathcal{A}]\!]_{\text{sem}}) \mid$ States$(\rho) \cap$ Bad $= \emptyset\}$. A strategy f_o is a *winning strategy* from state s if and only if, for all strategies f_i of input, Outcome$_o(s, f_o, f_i) \subseteq W^o(\text{Bad})$. On the contrary, a strategy f_i for input is a *spoiling strategy* of f_o if and only if Outcome$(s, f_o, f_i) \nsubseteq W^o(\text{Bad})$. A state s is winning for output if there exists a winning strategy from s. The game $(\mathcal{A}, W^o(\text{Bad}))$ is winning if and only if the initial state is winning. Solving this game is decidable [32,28,13]. We only consider safety games in this paper, and without lost of generality we assume these "bad" states are specified by a set of entirely "bad" locations (in the sense that all states in which such a location participates are bad).

Strategies in timed games as operators on timed specifications. We sketch how timed games can be used to establish consistency of a timed specification.

An *immediate error* occurs 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 err$^{\mathcal{S}} \subseteq St^{[\![\mathcal{S}]\!]_{\text{sem}}}$ as:

$$\text{err}^{\mathcal{S}} = \left\{ s \mid (\exists d.\ s \xrightarrow{d} \!\!\!\!\!\!/\ ) \text{ and } \forall d\ \forall o!\ \forall s'.\ s \xrightarrow{d} s' \text{ implies } s' \xrightarrow{o!} \!\!\!\!\!\!/\ \right\}$$

It follows that no immediate error states can occur in implementations, since they verify independent progress. In [13] we show that $\mathcal{S}$ is consistent (see Definition 6) iff there exists a winning strategy for output in the safety game $(\mathcal{S}, W^o(\text{err}^{\mathcal{S}}))$.

Consider the example specification of another coffee machine on Fig. 5. There is a unique reachable error state err$^{\mathcal{S}} = \{(\text{Blocked}, 0)\}$. Now let us take a strategy for the output player, f_o, such that $f_o((\text{Serving}, 4)) = \text{coff!}$, and $\forall y \neq 4. f_o((\text{Serving}, y)) = \text{delay}$. It can be translated into an implementation of the coffee machine, in which tea is never served. Thus this specification is consistent.
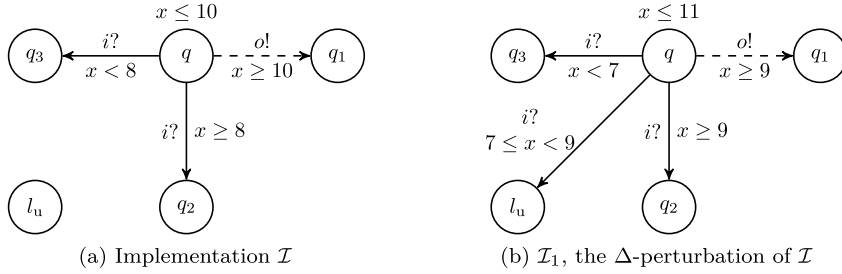
(a) Implementation $\mathcal{I}$ (b) $\mathcal{I}_1$, the Δ-perturbation of $\mathcal{I}$

Fig. 6. Illustration of the Δ-perturbation of an implementation $\mathcal{I}$ by $\Delta = 1$.

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 [21] 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 the constraint $\varphi \in \mathcal{B}(X)$ be a guard over the set of clocks X. For all $\Delta \in \mathbb{Q}_{\geqslant 0}$, the *enlarged guard* $\lceil \varphi \rceil_\Delta$ is constructed according to the following rules:

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

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

- Every edge $(q, o!, \varphi, \lambda, q') \in E$ is replaced by $(q, o!, \lceil \varphi \rceil_\Delta, \lambda, q') \in E'$,
- Every edge $(q, i?, \varphi, \lambda, q') \in E$ is replaced by $(q, i?, \lfloor \varphi \rfloor_\Delta, \lambda, q') \in E'$,
- $\forall q \in Loc. \, Inv'(q) = \lceil Inv(q) \rceil_\Delta$,
- $\forall q \in Loc. \, \forall i? \in Act_i$ there exists and edge $(q, i?, \varphi_u, \emptyset, l_u) \in E'$ with

$$\varphi_u = \neg \left(\bigvee_{(q, i?, \varphi, \lambda, q') \in E} \lfloor \varphi \rfloor_\Delta \right)$$

$\mathcal{I}_\Delta$ is not necessarily action deterministic, as output guards are enlarged. However it is input-enabled, since by construction (last case in the previous definition), any input not accepted after restricting input guards is redirected to the universal location l_u. Also $\mathcal{I}_0$ equals $\mathcal{I}$. An illustration of this transformation is presented in Fig. 6.

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 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 [21] 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 [21] 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 (Definition 4), while the perturbation of Definition 9 is monotonic.

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

Definition 10. An implementation $\mathcal{I}$ robustly satisfies a specification $\mathcal{S}$ for a given delay $\Delta \in \mathbb{Q}_{\geqslant 0}$, denoted $\mathcal{I}\ \mathbf{sat}_\Delta\ \mathcal{S}$, iff $\mathcal{I}_\Delta \leqslant \mathcal{S}$. We write $[\![\mathcal{S}]\!]^\Delta_{\mathrm{mod}}$ for the set of all Δ-robust implementations of $\mathcal{S}$, such that

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

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

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

Proof. 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 $[\![\mathcal{I}_{\Delta_1}]\!]_{\mathrm{sem}}$ and $[\![\mathcal{I}_{\Delta_2}]\!]_{\mathrm{sem}}$. Let $R = \{(s_1, s_2) \in St^{[\![\mathcal{I}_{\Delta_1}]\!]_{\mathrm{sem}}} \times St^{[\![\mathcal{I}_{\Delta_2}]\!]_{\mathrm{sem}}} \mid s_1 = (q, u) = s_2\}$ be a candidate alternating simulation relation. We prove by co-induction that R satisfies Definition 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 $d \in \mathbb{R}_{\geqslant 0}$, 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, similarly, $u' \models \lceil Inv(q')\rceil_{\Delta_1} \Rightarrow u' \models \lceil Inv(q')\rceil_{\Delta_2}$.
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 $i?$). Besides $u \models \varphi$ and therefore we assume that $u' \models Inv(q')$, which implies that $u' \models \lceil Inv(q')\rceil_{\Delta_1}$.

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

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

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

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

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

$$\mathcal{S} \leqslant \mathcal{T} \implies [\![\mathcal{S}]\!]^\Delta_{\mathrm{mod}} \subseteq [\![\mathcal{T}]\!]^\Delta_{\mathrm{mod}}$$

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

Definition 11. Let $\mathcal{S}$ be a specification and $\Delta \in \mathbb{Q}_{>0}$, then $\mathcal{S}$ is Δ-*robust consistent* iff 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 5. Fig. 7 presents a Δ-perturbation of the researcher implementation presented in Fig. 3. This implementation robustly satisfies the specification of Fig. 1(b) for any $\Delta \in\ ]0, 1]$. However, for $\Delta = 2$ the following run is possible in the perturbed implementation:

$$\big(\mathsf{Idle}, (0)\big) \xrightarrow{\mathrm{cof?}} \big(\mathsf{C}, (0)\big) \xrightarrow{5} \big(\mathsf{C}, (5)\big)$$

but it cannot be matched by the specification because it exceeds the invariant of the location corresponding to C in Fig. 1(b).

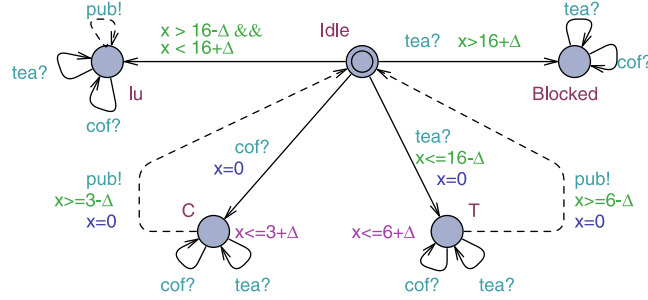**Fig. 7.** Δ-perturbation of the researcher implementation.

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. Note that it is not enough to restrict the specifications in order to synthesize robust components, since the behaviors removed might still happen after the Δ-perturbation and could lead to error states. However we propose a construction that encodes a robust game into a classical timed game.

De Alfaro et al. show [31] 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 7. 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 [28]. The following definition formalizes the notion of a *symbolic strategies*, which can be represented using symbolic states only:

Definition 12. A *symbolic strategy* F for the output player is a function $\mathcal{Z} \mapsto Act_{\mathrm{o}} \cup \{\mathsf{delay}\}$, where $\mathcal{Z}$ is a set of symbolic states, such that whenever $F((q, Z)) \in Act_{\mathrm{o}}$ 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 13. 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 $\mathcal{G}_\mathcal{A} = (\mathcal{R}_\mathcal{A}, \to^G)$ of $\mathcal{A}$. For each region r in location q, the incident edges and the invariant are defined as follows:

- $((q, r), \tau, r\nearrow, \emptyset, (q, r\nearrow)) \in \widehat{E}$ iff $(q, r) \xrightarrow{\tau}^G (q, r\nearrow)$.
- For each edge $(q, i?, \varphi, \lambda, q') \in E$, $((q, r), i?, \varphi, \lambda, (q', r')) \in \widehat{E}$, iff $(q, r) \xrightarrow{i?}^G (q', r')$.
- 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 for each edge $(q, o!, \varphi, \lambda, q') \in E$, $((q, r), o!, \varphi, \lambda, (q', r')) \in \widehat{E}$, iff $(q, r) \xrightarrow{o!}^G (q', r')$.

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 action can happen within Δ time of what the strategy originally prescribes. We write $\lceil \mathcal{A} \rceil_\Delta^{\mathrm{o}}$ (respectively $\lceil \mathcal{A} \rceil_\Delta^{\mathrm{i}}$) 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 14. 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.* $\mathsf{Runs}(\llbracket \lceil \mathcal{A}_F \rceil_\Delta^{\mathrm{o}} \rrbracket_{\mathrm{sem}}) \subseteq W^{\mathrm{o}}(\mathsf{Bad})$.

In the rest of this section we describe a technique to find these robust strategies by modifying the original game automaton.
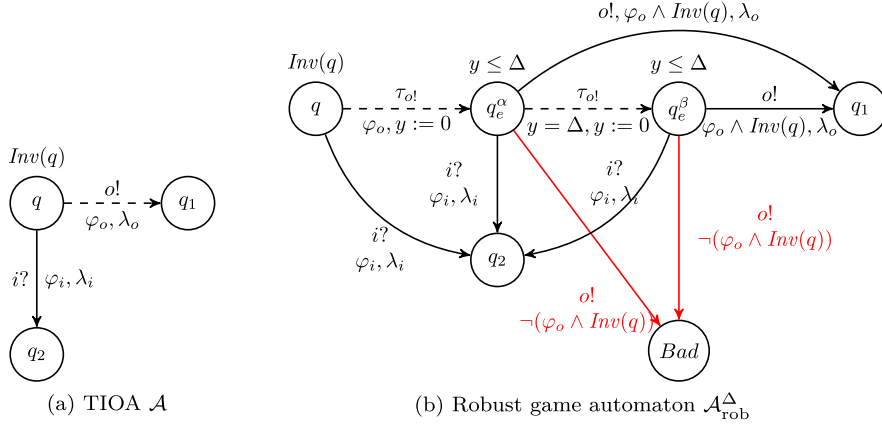
Fig. 8. Illustration of the construction of the robust game automaton $\mathcal{A}_{\text{rob}}^{\Delta}$, from an automaton $\mathcal{A}$, with one location q and incident edges $e = (q, o!, \varphi_o, \lambda_o, q_1)$ and $(q, i?, \varphi_i, \lambda_i, q_2)$.

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 [16]. Below in Definition 15, we propose an extended version of the construction presented in [16]. We admit both positive and negative perturbations of the player moves. In the original construction of [16] 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 Definition 14.

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

Definition 15. The *robust game automaton* $\mathcal{A}_{\text{rob}}^{\Delta} = (\widetilde{Loc}, q_0, Clk \cup \{y\}, \widetilde{E}, \widetilde{Act}, \widetilde{Inv})$ is constructed from $\mathcal{A}$, with an additional clock y, input actions $\widetilde{Act_i} = Act_i \cup Act_o$, and output actions $\widetilde{Act_o} = \{\tau_{o!} \mid o! \in Act_o\}$, according to the following rules. For each location $q \in Loc$, and for each edge $e = (q, o!, \varphi, \lambda, q') \in E$:

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

Actions $\tau_{o!}$ are considered as silent actions, and consequently they will be concealed from the runs of the automaton.

Technically speaking, since in a TIOA guards must be convex, the two transitions to the Bad location (drawn in red on Fig. 8) may be split into several copies, one for each convex guard in $\neg\varphi$.

The construction is illustrated in Fig. 8. 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 forbids output to play any action with a reaction time smaller than Δ. More precisely, any strategy for output found in the robust game automaton $\mathcal{A}_{\text{rob}}^{\Delta}$ will effectively correspond to a non-zeno strategy in $\mathcal{A}$.

2. Input can perturb this move with $d' \leqslant \Delta$, such that action $o!$ is performed with either a smaller delay:

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

or a greater delay:

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

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

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

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

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 $(\mathcal{A}_{\text{rob}}^\Delta, W^o(\text{Bad}))$, then F_{rob} (constructed above) is a Δ-robust winning strategy for output in the game $(\mathcal{A}, W^o(\text{Bad}))$.*

Proof. We consider the automaton $\mathcal{A}_{F_{\text{rob}}}$ representing the outcome of applying F_{rob} to $\mathcal{A}$. We must prove that $\text{Runs}(\llbracket \lceil \mathcal{A}_{F_{\text{rob}}} \rceil_\Delta^o \rrbracket_{\text{sem}}) \subseteq W^o(\text{Bad})$, where $W^o(\text{Bad}) = \{\rho \in \text{Runs}(\llbracket \mathcal{A} \rrbracket_{\text{sem}}) \mid \text{States}(\rho) \cap \text{Bad} = \emptyset\}$.

First, we map each run $\widehat{\rho}$ of $\lceil \mathcal{A}_{F_{\text{rob}}} \rceil_\Delta^o$ 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 $\text{Runs}(\llbracket \lceil \mathcal{A}_{F_{\text{rob}}} \rceil_\Delta^o \rrbracket_{\text{sem}})$ such that $\text{last}(\widehat{\rho}_i) = ((q_i, r_i), u_i)$, then there exists a run ρ_i in $\text{Runs}(\llbracket \mathcal{A} \rrbracket_{\text{sem}})$, such that $\text{last}(\rho_i) = (q_i, u_i)$.

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 $F(q_i^\alpha, \widetilde{r}_i)) = a$. Since $u_i \models \lceil r_i \rceil_\Delta$, $\exists \delta \in [-\Delta, \Delta]. u_i + \delta \models r_i$. 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 $\mathcal{A}_{\text{rob}}^\Delta$, F is a winning strategy, which implies that $\forall \delta' \in [-\Delta, \Delta]. \widetilde{u}_{i\delta} + \delta' \models \varphi \wedge Inv(q_i)$ (otherwise input as a spoiling strategy).
   This proves that $u_i \models \varphi \wedge Inv(q_i)$ 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 $a \in \mathbb{R}_{\geqslant 0}$, 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 $\rho_{i+1} \in \text{Runs}(\llbracket \mathcal{A} \rrbracket_{\text{sem}})$.
   Otherwise output has a strategy that eventually performs an action. This is represented by the following sequence of edges in $\mathcal{A}_{F_{\text{rob}}}$: $(q_i, r_i) \xrightarrow{\tau} (q_i, r_{i1}) \xrightarrow{\tau} (q_i, r_{i2}) \ldots (q_i, r_{in}) \xrightarrow{o!} (q_i', r_i')$. We consider that a is the maximum delay firable from $\widehat{\rho}_i$, thus $\widehat{\rho}_{i+1} = \widehat{\rho}_i \xrightarrow{a_1} ((q_i, r_{i1}), u_i + a_1) \xrightarrow{\tau} ((q_i, r_{i2}), u_i + a_1) \xrightarrow{a_2} ((q_i, r_{i2}), u_i + a_1 + a_2) \xrightarrow{\tau} \ldots ((q_i, r_{in}), u_i + a) \xrightarrow{o!} ((q_i', r_i'), u_i')$, such that $a = \sum_{j=1}^n a_j$. Then $u_i + a \models \lceil r_{in} \rceil_\Delta$. and as in the previous case we can prove that $u_i + a \models \varphi \wedge Inv(q_i)$. This proves that $\rho_{i+1} \in \text{Runs}(\llbracket \mathcal{A} \rrbracket_{\text{sem}})$.

We now prove that all the states in $\lceil \mathcal{A}_{F_{\text{rob}}} \rceil_\Delta^o$ are safe, by mapping the runs of $\mathcal{A}_{F_{\text{rob}}}$ with the ones of $\mathcal{A}_{\text{rob}}^\Delta$. We use another induction on the length on the run. Let assume that $\widehat{\rho}_i \in \text{Runs}(\llbracket \lceil \mathcal{A}_{F_{\text{rob}}} \rceil_\Delta^o \rrbracket_{\text{sem}})$ is mapped to a run $\widetilde{\rho}_i \in \text{Runs}(\llbracket \mathcal{A}_{\text{rob}}^\Delta \rrbracket_{\text{sem}})$, such that if $\text{last}(\widehat{\rho}_i) = ((q_i, r_i), u_i)$, then $\text{last}(\widetilde{\rho}_i) = (q_i, \widetilde{u}_i)$, $u_i = \widetilde{u}_{i|Clk}$, 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 $\mathcal{A}_{\text{rob}}^\Delta$ without reaching an unsafe state. For any delay $d \in \mathbb{R}_{\geqslant 0}$, 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 $\mathcal{A}_{\text{rob}}^\Delta$: $(q_i, \tau_{o!}, \varphi, \lambda \cup \{y\}, q_{i_e}^\alpha)$, $(q_i, \tau_{o!}, \{y = \Delta\}, \{y\}, q_{i_e}^\beta)$, $(q_{i_e}^\alpha, o!, \{y \leqslant \Delta\}, \{y\}, q_{i+1})$ and $(q_{i_e}^\beta, o!, \{y \leqslant \Delta\}, \{y\}, q_{i+1})$. 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 $u_i + a \models \lceil r_i' \rceil_\Delta$.
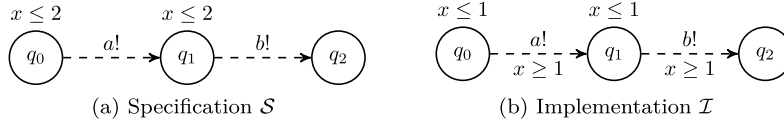
(a) Specification $\mathcal{S}$ (b) Implementation $\mathcal{I}$

Fig. 9. Specification with 1-robust implementation, but with no robust strategy in the robust game automaton.

Then by construction of F_{rob}, $F(q_{i_e}^{\alpha}, \widetilde{r}_i') = \tau_{o!}$, and there exists $b \in \mathbb{R}_{\geqslant 0}$ 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, $\forall \delta \in [-\Delta, \Delta].(q_i, \widetilde{u}_i) \xrightarrow{b+\delta} (q_i, \widetilde{u}_i + b + \delta) \xrightarrow{o!}$. This is in particular the case for $b + \delta = a$ since $a \in \lceil \Delta \rceil_{[} r_i'$. 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 exist specifications that can be robustly implemented, although they will be judged as non-robust using the robust game automaton construction. For instance, the specification $\mathcal{S}$ in Fig. 9(a) is 1-robust consistent and it can be robustly implemented with the implementation $\mathcal{I}$ in Fig. 9(b) (the Δ-perturbation $\mathcal{I}_1$ corresponds to the same TIOA as $\mathcal{S}$). But for $\Delta = 1$ no robust strategy exists in the robust game automaton. Indeed, since clock x is not reset by the edge from q_0 to q_1, the following run is possible in the semantics of $\mathcal{I}_1$: $(q_0, 0) \xrightarrow{2}^{\mathcal{I}} (q_0, 2) \xrightarrow{a!}^{\mathcal{I}} (q_1, 2) \xrightarrow{b!}^{\mathcal{I}} (q_2, 2)$. Therefore in this run action $b!$ must happen immediately. This is not robust according to the robust game automaton, as it cannot be perturbed.

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 the 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 $\mathrm{err}^{\mathcal{S}}$. 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 actions trigger 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) satisfies 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 16. For a specification $\mathcal{S}$ and $\Delta \in \mathbb{R}_{\geqslant 0}$, the set $\mathrm{err}_{\Delta}^{\mathcal{S}}$ of error states for Δ-robust consistency is such that $(q, u) \in \mathrm{err}_{\Delta}^{\mathcal{S}}$ iff one the following conditions is verified:

- Violates independent progress: $(\exists d \in \mathbb{R}_{\geqslant 0}. (q, u) \xrightarrow{d}{\nrightarrow})$ and $(\forall d. \forall o!. (q, u) \xrightarrow{d} (q, u + d) \Rightarrow (q, u + d) \xrightarrow{o!}{\nrightarrow})$.
- Violates Δ-latency of inputs: $\exists e = (q, i?, \varphi, c, q')$, with $q' \neq l_{\mathrm{u}}$, enabled in (q, u), such that $\exists d \in [0, 2\Delta]. (q, u) \xrightarrow{d} (q, u + d)$ 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 $\mathrm{err}^{\mathcal{S}} \subseteq \mathrm{err}_{\Delta}^{\mathcal{S}}$, because the error condition with robustness is weaker than in the classical case (cf. p. 103).

The Δ-robust consistency game $(\mathcal{S}, W^{\mathrm{o}}(\mathrm{err}_{\Delta}^{\mathcal{S}}))$ can be solved using the construction of Definition 15. 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 enforce 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 $q^* \neq l_u$), and for each location (q', r') that can be reach from (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 $q' \neq l_u$ 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. $\mathcal{S}$ is Δ-robust consistent if it admits a Δ-robust implementation. $\mathcal{I}_F$ satisfies 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

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

Note that since F is a robust winning strategy, the runs of $\lceil \mathcal{S}_F \rceil_\Delta^o$ (and by construction the ones of $(\mathcal{I}_F)_\Delta$) also belong to $\mathcal{S}$. Finally we assume that $\mathcal{S}$ can accept τ transitions in any state 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 $d \in \mathbb{R}_{\geqslant 0}$, 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 an 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 an 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 [13]:

Definition 17. 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 according to the following rule:

$$\left((q_s, q_t), a, \varphi_s \wedge \varphi_t, \lambda_s \cup \lambda_t, (q'_s, q'_t) \right) \in E \quad \text{iff} \quad \left(q_s, a, \varphi_s, \lambda_s, q'_s \right) \in E^S \text{ and } \left(q_t, a, \varphi_t, \lambda_t, q'_t \right) \in E^T$$

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 $\Delta \in \mathbb{Q}_{>0}$:*

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

Proof. The theorem is a direct extension of Theorem 6 in [13], but now for robust implementations. By definition of the robust implementation,
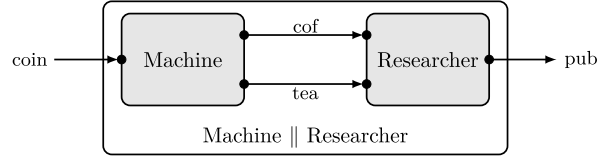
Fig. 10. Diagram for the parallel composition of a coffee machine and a researcher.

$$\mathcal{I} \in [\![\mathcal{S} \wedge \mathcal{T}]\!]^\Delta_{\mathrm{mod}} \quad \Leftrightarrow \quad \mathcal{I}_\Delta \leqslant \mathcal{S} \wedge \mathcal{T}$$

According to Theorem 6 in [13], items 1 and 2,

$$\mathcal{I}_\Delta \leqslant \mathcal{S} \wedge \mathcal{T} \quad \Leftrightarrow \quad \mathcal{I}_\Delta \leqslant \mathcal{S} \wedge \mathcal{I}_\Delta \leqslant \mathcal{T}$$

And the last terms correspond to the definition of robust implementations. □

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:

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

So requirements with different precision can be conjoined, by considering the smaller jitter. Robustness of the operator in Definition 17 is very fortunate. Thanks to this, large parts of implementation of theory of [13] 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 $Act_o^\mathcal{S} \cap Act_o^\mathcal{T} = \emptyset$. Parallel composition is obtained in [13] by a product, where the inputs of one specification synchronize with the outputs of the other:

Definition 18. 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 composition, denoted $\mathcal{S} \parallel \mathcal{T}$, as the TIOA $(Loc, q_0, Clk, E, Act, Inv)$, where $Loc = Loc^S \times Loc^T$, $q_0 = (q_0^S, q_0^T)$, $Clk = Clk^S \uplus Clk^T$, $Act = Act_o \cup Act_i$ with $Act_o = Act_o^S \uplus Act_o^T$ and $Act_i = (Act_i^S \setminus Act_o^T) \cup (Act_i^T \setminus Act_o^S)$, $Inv(q_s, q_t) = Inv(q_s) \wedge Inv(q_t)$, and the set of edges is defined by the three following rules:

- Let $a \in Act^S \setminus Act^T$, for each $q_t \in Loc^T$, $((q_s, q_t), a, \varphi_s, \lambda_s, (q'_s, q_t)) \in E$, iff $(q_s, a, \varphi_s, \lambda_s, q'_s) \in E^S$.
- Let $a \in Act^T \setminus Act^S$, for each $q_s \in Loc^S$, $((q_s, q_t), a, \varphi_t, \lambda_t, (q_s, q'_t)) \in E$, iff $(q_t, a, \varphi_t, \lambda_t, q'_t) \in E^T$.
- Let $a \in Act^S \cap Act^T$, $((q_s, q_t), a, \varphi_s \wedge \varphi_t, \lambda_s \cup \lambda_t, (q''_s, q'_t)) \in E$ iff $(q_s, a, \varphi_s, \lambda_s, q'_s) \in E^S$ and $(q_t, a, \varphi_t, \lambda_t, q'_t) \in E^T$.

We also recall the definition of the parallel product for two TIOTS $S = (St^S, s_0, \Sigma^S, \to^S)$ and $T = (St^T, t_0, \Sigma^T, \to^T)$. $S \otimes T = (St^S \times St^T, (s_0, t_0), \Sigma^{S \otimes T}, \to^{S \otimes T})$, such that:

$$\frac{s \xrightarrow{a}^S s' \quad a \in \Sigma^S \setminus \Sigma^T}{(s, t) \xrightarrow{a}^{S \otimes T} (s', t)} \text{ indep-l} \qquad \frac{t \xrightarrow{a}^T t' \quad a \in \Sigma^T \setminus \Sigma^S}{(s, t) \xrightarrow{a}^{S \otimes T} (s, t')} \text{ indep-r}$$

$$\frac{s \xrightarrow{a}^S s' \quad t \xrightarrow{a}^T t' \quad a \in \mathbb{R}_{\geqslant 0} \cup \Sigma_i^{S \otimes T} \cup (\Sigma_i^S \cap \Sigma_o^T) \cup (\Sigma_o^S \cap \Sigma_i^T)}{(s, t) \xrightarrow{a}^{S \otimes T} (s', t')} \text{ sync}$$

Example 6. The two timed specifications in Fig. 1(a) and 1(b) 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. 10.

In the input-enableness setting we model incompatibility by introducing a predicate describing undesirable states, here denoted by a set und. It should in general contain the universal location l_u. 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 [14] that is *two specifications can be composed if there exists at least one environment 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 19. A specification $\mathcal{S}$ is Δ-*robust useful* if there exists an environment $\mathcal{E}$ such that no undesirable states are reached in $[\![\lceil\mathcal{E}\rceil^o_\Delta \parallel \mathcal{S}]\!]_{\text{sem}}$.

Remark that, contrary to robust implementations, we only apply a perturbation to the outputs of the environment. Indeed since the environment is a complement of the system, its inputs correspond to the output of the system, and therefore they are not perturbed to achieve robust compatibility.

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

Proof. If $\mathcal{S}$ is 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 $\Delta_1 \leqslant \Delta_2$, $[\![\lceil\mathcal{E}\rceil^o_{\Delta_1}]\!]_{\text{sem}} \subseteq [\![\lceil\mathcal{E}\rceil^o_{\Delta_2}]\!]_{\text{sem}}$. 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. $\quad\square$

To check robust usefulness we solve the robust game $(\mathcal{S}, W^i(\text{und}))$, 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. 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 composition. 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. $\quad\square$

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

Definition 20. 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 [13] remain valid. Moreover, robustness distributes over parallel composition in the following fashion:

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

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

Proof. In the following, we denote by $(Loc^k, q_0^k, Clk^k, E^k, Act^k, Inv^k)$, with $k \in \{I, J, I \parallel J\}$, the TIOAs corresponding to $\mathcal{I}, \mathcal{J}$, or $\mathcal{I} \parallel \mathcal{J}$, respectively, and by $(St^k, (q_0^k, 0), Act^k, \to^k)$, 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 [13] that states that $[\![\mathcal{I}_\Delta \parallel \mathcal{J}_\Delta]\!]_{\text{sem}} = [\![\mathcal{I}_\Delta]\!]_{\text{sem}} \otimes [\![\mathcal{I}_\Delta]\!]_{\text{sem}}$. Then we need to prove the refinement $[\![(\mathcal{I} \parallel \mathcal{J})_\Delta]\!]_{\text{sem}} \leqslant [\![\mathcal{I}_\Delta]\!]_{\text{sem}} \otimes [\![\mathcal{J}_\Delta]\!]_{\text{sem}}$ by witnessing the following relation:

$$R = \big\{ \big(((q_i, q_j), u_{ij}), ((\widehat{q}_i, u_i), (\widehat{q}_j, u_j))\big) \in St^{[I\|J]_\Delta} \times \big(St^{I_\Delta} \times St^{J_\Delta}\big) \mid$$
$$\big((q_i = \widehat{q}_i \wedge u_i = u_{ij|Clk^I}) \vee \widehat{q}_i = l_u\big) \wedge \big((q_j = \widehat{q}_j \wedge u_j = u_{ij|Clk^J}) \vee \widehat{q}_j = l_u\big)\big\}$$

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 $((q_i, q_j), u_{ij}) \xrightarrow{d} {}^{[I\|J]_\Delta} ((q_i, q_j), u_{ij} + d)$ for some $d \in \mathbb{R}_{\geqslant 0}$, then by definition $u_{ij} \models \lceil Inv(q_i, q_j)\rceil_\Delta$. By construction of the parallel composition $\lceil Inv(q_i, q_j)\rceil_\Delta = \lceil Inv(q_i) \wedge Inv(q_j)\rceil_\Delta = \lceil Inv(q_i)\rceil_\Delta \wedge \lceil Inv(q_j)\rceil_\Delta$. 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 $(l_u, u) \xrightarrow{d} (l_u, u + d)$.
   By definition of the sync rule from $[\![\mathcal{I}_\Delta]\!]_{\text{sem}} \otimes [\![\mathcal{J}_\Delta]\!]_{\text{sem}}$,

   $$\big((\widehat{q}_i, u_i), (\widehat{q}_j, u_j)\big) \xrightarrow{d} {}^{I_\Delta \otimes J_\Delta} \big((\widehat{q}_i, u_i + d), (\widehat{q}_j, u_j + d)\big)$$

   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 $\exists e \in E^{I \| J}. e = ((q_i, q_j), o!, \varphi, c, (q_i', q_j'))$ such that $u_{ij} \models \lceil \varphi \rceil_\Delta$ and $u_{ij}' \models \lceil Inv(q_i', q_j') \rceil_\Delta$,

   2.1. 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 $u_{ij|Clk^I}' = u_j$. 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, $(l_u, u_i) \xrightarrow{o!} {}^{I_\Delta} (l_u, u_i)$. Then, according to the indep-I rule, $((\hat{q}_i, u_i), (\hat{q}_j, u_j)) \xrightarrow{o!} {}^{I_\Delta \otimes J_\Delta} ((\hat{q}_i', u_i'), (\hat{q}_j, u_j))$. Moreover, either $\hat{q}_i = q_i$ and then $\hat{q}_i' = q_i'$, or $\hat{q}_i = l_u = \hat{q}_i'$, which proves that the relation R is preserved.

   2.2. If $o \in Act_o^J \cap Act_i^I$ (the reverse case $o \in Act_o^I \cap Act_i^J$ 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 edge $e_j' = (q_j, o?, \varphi_j', c_j', q_j'') \in E^J$ such that $u_j \models \lfloor \varphi_j' \rfloor_\Delta$ (since $u_j \models \lceil \varphi_j \rceil_\Delta$ it implies that $\exists \varepsilon \in [-\Delta, \Delta]. u_j + \varepsilon \models \varphi_j$, but in the same time it would be the case that $u_j + \varepsilon \models \varphi_j'$. This is a contradiction 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 $(q_j, u_j) \xrightarrow{o?} {}^{J_\Delta} (l_u, u_j)$.

   Then, according to the sync rule, $((\hat{q}_i, u_i), (\hat{q}_j, u_j)) \xrightarrow{o!} {}^{I_\Delta \otimes J_\Delta} ((\hat{q}_i', u_i'), (\hat{q}_j', u_j'))$, and as previously, by construction we check that the relation is preserved.

3. Finally, if $((\hat{q}_i, \hat{u}_i), (\hat{q}_j, \hat{u}_j)) \xrightarrow{i?} {}^{I_\Delta \otimes J_\Delta} ((\hat{q}_i', u_i'), (\hat{q}_j', u_j'))$, 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:

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

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

Finally, we show in Theorem 5 that the independent implementability property of [13] can be extended to robust implementability, which follows from Lemma 1 and Theorem 10 in [13].

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. $\mathcal{I}$ **sat**$_\Delta$ $\mathcal{S}$ and $\mathcal{J}$ **sat**$_\Delta$ $\mathcal{S}$. Then $\mathcal{I} \| \mathcal{J}$ **sat**$_\Delta$ $\mathcal{S} \| \mathcal{T}$. Moreover if $\mathcal{S}$ and $\mathcal{T}$ are Δ-compatible then $\mathcal{I}$ and $\mathcal{J}$ are also Δ-compatible.*

Proof. The first part of the theorem is deduced from previous results:

- Due to Lemma 1, $[\![(\mathcal{I} \| \mathcal{J})_\Delta]\!]_{\text{sem}} \leqslant [\![\mathcal{I}_\Delta]\!]_{\text{sem}} \otimes [\![\mathcal{J}_\Delta]\!]_{\text{sem}}$.
- Then, Theorem 10 from [13] proves that refinement is a pre-congruence with respect to parallel composition: if $S_1 \leqslant S_2$ and T is composable with S_1, then $S_1 \otimes T \leqslant S_2 \otimes T$. Observe that since $\mathcal{S}$ and $\mathcal{T}$ are composable, so are $\mathcal{I}$ and $\mathcal{J}$, and their semantics. Thus we can apply it twice:
  $[\![\mathcal{I}_\Delta]\!]_{\text{sem}} \leqslant [\![\mathcal{S}]\!]_{\text{sem}} \Rightarrow [\![\mathcal{I}_\Delta]\!]_{\text{sem}} \otimes [\![\mathcal{J}_\Delta]\!]_{\text{sem}} \leqslant [\![\mathcal{S}]\!]_{\text{sem}} \otimes [\![\mathcal{J}_\Delta]\!]_{\text{sem}}$, and
  $[\![\mathcal{J}_\Delta]\!]_{\text{sem}} \leqslant [\![\mathcal{T}]\!]_{\text{sem}} \Rightarrow [\![\mathcal{S}]\!]_{\text{sem}} \otimes [\![\mathcal{J}_\Delta]\!]_{\text{sem}} \leqslant [\![\mathcal{S}]\!]_{\text{sem}} \otimes [\![\mathcal{T}]\!]_{\text{sem}}$.
- Finally Theorem 11 in [13] allows to lift the result to the composition of TIOA specifications: $[\![\mathcal{S}]\!]_{\text{sem}} \otimes [\![\mathcal{T}]\!]_{\text{sem}} = [\![\mathcal{S} \| \mathcal{T}]\!]_{\text{sem}}$.

Due to the transitivity of the refinement, we can concatenate the previous results, which proves that $[\![(\mathcal{I} \| \mathcal{J})_\Delta]\!]_{\text{sem}} \leqslant [\![\mathcal{S} \| \mathcal{T}]\!]_{\text{sem}}$, and therefore $\mathcal{I} \| \mathcal{J}$ **sat**$_\Delta$ $\mathcal{S} \| \mathcal{T}$.

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 $\lceil \mathcal{E} \rceil_\Delta^0 \| (\mathcal{S} \| \mathcal{T})$ avoids reaching any undesirable states. From Theorems 10 and 11 in [13], we get that $\mathcal{I} \| \mathcal{J} \leqslant \mathcal{S} \| \mathcal{T}$. Again with Theorem 10 we get that $(\mathcal{I} \| \mathcal{J}) \| \lceil \mathcal{E} \rceil_\Delta^0 \leqslant (\mathcal{S} \| \mathcal{T}) \| \lceil \mathcal{E} \rceil_\Delta^0$. Consequently, since no undesirable states are reached in $(\mathcal{S} \| \mathcal{T}) \| \lceil \mathcal{E} \rceil_\Delta^0$ this is also the case in $(\mathcal{I} \| \mathcal{J}) \| \lceil \mathcal{E} \rceil_\Delta^0$, which proves that $\mathcal{I} \| \mathcal{J}$ is Δ-useful and so $\mathcal{I}$ and $\mathcal{J}$ are Δ-compatible. $\square$

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

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

5.4. Quotient

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

To apply quotienting, we require that $Act^S \subseteq Act^T$ and $Act_o^S \subseteq Act_o^T$. 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 21. Let $S = (Loc^S, q_0^S, Clk^S, E^S, Act^S, Inv^S)$ and $T = (Loc^T, q_0^T, Clk^T, E^T, Act^T, Inv^T)$ be two specifications, with $Act^S \subseteq Act^T$ and $Act_o^S \subseteq Act_o^T$.

Their *quotient*, denoted $T \backslash\!\backslash S$, is the TIOA $(Loc, q_0, Clk, E, Act, Inv)$ where $Loc = Loc^T \times Loc^S \cup \{l_u, l_\emptyset\}$, $q_0 = (q_0^T, q_0^S)$, $Clk = Clk^T \uplus Clk^S \uplus \{x_{new}\}$, $Act = Act_i \uplus Act_o$ with $Act_i = Act_i^T \cup Act_o^S \cup \{i_{new}\}$ and $Act_o = Act_o^T \setminus Act_o^S$, $Inv(q_t, q_s) = Inv(l_u) = true$ and $Inv(l_\emptyset) = \{x_{new} \leqslant 0\}$, and the set E of edges is defined by the following rules:

- $((q_t, q_s), a, \neg Inv^S(q_s), \{x_{new}\}, l_u) \in E$ iff $q_t \in Loc^T, q_s \in Loc^S, a \in Act$.
- $((q_t, q_s), i_{new}, \neg Inv(q_t) \wedge Inv(q_s), \{x_{new}\}, l_\emptyset) \in E$ iff $q_t \in Loc^T, q_s \in Loc^S$.
- $((q_t, q_s), a, \varphi^T \wedge \varphi^S, \lambda_t \cup \lambda_s, (q_t', q_s')) \in E$ iff $(q_t, a, \varphi_t, \lambda_t, q_t') \in E^T$ and $(q_s, a, \varphi_s, \lambda_s, q_s') \in E^S$.
- Let $a \in Act_o^S$ and $G^T = \bigvee \{\varphi_t \mid (q_t, a, \varphi_t, \lambda_t, q_t') \in E^T\}$, $((q_t, q_s), a, \varphi^S \wedge \neg G^T, \{x_{new}\}, l_\emptyset) \in E$ iff $(q_s, a, \varphi_s, \lambda_s, q_s') \in E^S$.
- Let $a \notin Act^S$, $((q_t, q_s), a, \varphi^T, \lambda_t, (q_t', q_s')) \in E$ iff $\forall (q_t, a, \varphi_t, \lambda_t, q_t') \in E^T$.
- Let $a \in Act_o^S$ and $G^S = \bigvee \{\varphi_s \mid (q_s, a, \varphi_s, \lambda_s, q_s') \in E^S\}$, $((q_t, q_s), a, \neg G^S, \{\}, l_u) \in E$ iff $(q_t, a, \varphi_t, \lambda_t, q_t') \in E^T$.
- $(l_\emptyset, a, x_{new} = 0, \emptyset, l_\emptyset) \in E$ iff $a \in Act_i$.
- $(l_u, a, true, \emptyset, l_u) \in E$ iff $a \in Act$.

As stated in Theorem 12 of [13], the quotient gives a maximal (the weakest) specification for a missing component. This theorem can be generalized to specifications that are locally consistent (see [13]), 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 S and T be two specifications such that the quotient $T \backslash\!\backslash S$ is defined and let J be an implementation, then*:

$$S \parallel J_\Delta \leqslant T \quad iff \quad J \; \mathbf{sat}_\Delta \; T \backslash\!\backslash S$$

Proof. First let remark that J_Δ is a locally consistent specification, as defined in [13]. Then, we can apply Theorem 12 of [13] to J_Δ which proves:

$$S \parallel J_\Delta \leqslant T \quad iff \quad J_\Delta \leqslant T \backslash\!\backslash S$$

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. Now we will propose a technique that evaluates the greatest possible value of the perturbation. We follow a counterexample refinement approach, a technique used for automatic abstraction refinement in [33]. 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 S, determine the greatest value of Δ such that S is Δ-robust consistent.

Robust usefulness. Given a specification S, determine the greatest value of Δ such that S is Δ-robust useful.

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 [34,35]. We denote by $\mathcal{B}_\Delta(Clk)$ the set of parametric guards with parameter Δ over a set of clocks *Clk*. Parametric guards in $\mathcal{B}_\Delta(Clk)$ are generated by the following grammar $\varphi ::= x \prec l \mid x - y \prec l \mid \varphi \wedge \varphi$, where $x, y \in Clk$, $\prec \in \{<, \leqslant, >, \geqslant\}$ and $l = a + b * \Delta$ is a linear expression such that $a, b \in \mathbb{Q}$.

Definition 22. A *Parametric TIOA* with parameter Δ, is a TIOA $\mathcal{A}$ such that guards and invariants are replaced by parametric guards.

If $\mathcal{A}$ is a parametric TIOA and $W^o(\text{Bad})$ is a safety objectives, then $(\mathcal{A}, W^o(\text{Bad}))$ is parametric timed game. For a given value $\delta \in \mathbb{Q}$ of the parameter, we define the non-parametric TIOA $\mathcal{A}^\delta$ 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 triples (q, u, δ), where δ is a valuation 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 $X \nearrow^P$, $X \swarrow^P$, $\text{PPost}_a(X)$, $\text{PPred}_a(X)$ and $\text{PPred}_t(X, Y)$ stands for the extensions of previously defined non-parametric operations. Formally:

$$X \nearrow^P = \left\{ (q, u + d, \delta) \mid (q, u, \delta) \in X, \, d \in \mathbb{R}_{\geqslant 0} \right\}$$

$$X \swarrow^P = \left\{ (q, u - d, \delta) \mid (q, u, \delta) \in X, \, d \in \mathbb{R}_{\geqslant 0} \right\}$$

$$\text{PPost}_a(X) = \left\{ (q', u', \delta) \mid \exists (q, u, \delta) \in X. (q, u) \xrightarrow{a}^{\mathcal{A}_\delta} (q', u') \right\}$$

$$\text{PPred}_a(X) = \left\{ (q, u, \delta) \mid \exists (q', u', \delta) \in X. (q, u) \xrightarrow{a}^{\mathcal{A}_\delta} (q', u') \right\}$$

$$\text{PPred}_t(X, Y) = \left\{ (q, u, \delta) \mid \exists d \in \mathbb{R}_{\geqslant 0}. (q, u) \xrightarrow{d}^{\mathcal{A}_\delta} (q, u + d) \text{ and } (q, u + d) \in X \text{ and } \forall d' \in [0, d]. (q, u + d', \delta) \notin Y \right\}$$

6.2. Parametric robustness evaluation

Let $(\mathcal{A}_{\text{rob}}^\Delta, W^o(\text{Bad}))$ be a parametric timed game that solve a robustness problem (either robust consistency or robust usefulness). We define $\Delta_{max} = \text{Sup}\{\Delta \mid (\mathcal{A}_{\text{rob}}^\Delta, W^o(\text{Bad})) \text{ has a winning strategy}\}$. Computing Δ_{max} 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 [34]. 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 $(\mathcal{A}_{\text{rob}}^\Delta, W^o(\text{Bad}))$ for output (again it applies symmetrically to input).

Algorithm 1: Evaluation of the maximum robustness.

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

1 **begin**
2     $\Delta_{good} \leftarrow 0$
3     $\Delta_{bad} \leftarrow \Delta_{init}$
4   **while** $\Delta_{bad} - \Delta_{good} > \epsilon$ **do**
5         $(\Delta_{good}, \Delta_{bad}) \leftarrow \text{RefineValues}(\mathcal{A}_{\text{rob}}^\Delta, \Delta_{good}, \Delta_{bad})$
6   **end**
7   **return** Δ_{good}
8 **end**

The algorithm assumes that the game $(\mathcal{A}_{\text{rob}}^0, W^o(\text{Bad}))$ is won, and on contrary that $(\mathcal{A}_{\text{rob}}^{\Delta_{init}}, W^o(\text{Bad}))$ is lost. If Δ_{max} is not infinite, then the maximum constant in the automaton can be used for Δ_{init}. At the heart of the algorithm the procedure RefineValues solves the game $(\mathcal{A}_{\text{rob}}^\delta, W^o(\text{Bad}))$ for a value $\delta \in [\Delta_{good}, \Delta_{bad}]$. It updates 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 solves the game $(\mathcal{A}_{rob}^{\Delta_{mid}}, W^o(\text{Bad}))$. 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 an upper 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. Solve the game $(\mathcal{A}_{rob}^{\Delta_{bad}}, W^o(\text{Bad}))$.
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 the value $\Delta_{min} = \text{Inf}\{\Delta \mid F_i$ is a spoiling strategy in $(\mathcal{A}_{rob}^{\Delta}, W^o(\text{Bad}))\}$.
5. If Δ_{min} is not a minimum (F_i is not a spoiling strategy in $(\mathcal{A}_{rob}^{\Delta_{min}}, W^o(\text{Bad}))$) 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 $\mathcal{A}_{rob}^{\Delta}$, the symbolic graph $(\mathcal{Z}_{\mathcal{A}}^{\Delta_{bad}}, X_0, \to)$ computed for the game $(\mathcal{A}_{rob}^{\Delta_{bad}}, W^o(\text{Bad}))$, 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 $(\mathcal{A}_{rob}^{\Delta_{bad}}, W^o(\text{Bad}))$.

The algorithm is similar to the timed game algorithm proposed in [28] and implemented in the tool TIGA [36]. 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 a symbolic spoiling strategy F_i for input. First, for a symbolic state $X \in \mathcal{Z}_{\mathcal{A}}^{\Delta_{bad}}$, we define its timed successors restricted by the symbolic strategy F_i as:

$$X \nearrow^{F_i} = \big\{(q, u + d) \mid (q, u) \in X, d \in \mathbb{R}_{\geqslant 0}, \forall d' \in [0, d].$$

$$\text{if } \exists (q, Z) \in \mathcal{Z}_{\mathcal{A}}^{\Delta_{bad}}. \exists (q, Z') \in \mathcal{Z}_{\mathcal{A}}^{\Delta_{bad}}. u + d \in Z. u + d' \in Z'$$

$$\text{then } F_i\big((q, Z')\big) = F_i\big((q, Z)\big) \vee F_i\big((q, Z')\big) = \text{delay}\big\}$$

$X \nearrow^{F_i}$ is computed by taking the intersection of the timed successors of X with the symbolic states on which is defined the strategy. Outcome(F_i) is the subset of runs in the symbolic graph defined inductively by:

- $(q_0, X_0 \nearrow^{F_i}) \in \text{Outcome}(F_i)$,
- if $\rho \in \text{Outcome}(F_i)$ and $\text{last}(\rho) = (q, Z)$, then $\rho' = \rho \to (q', Z') \in \text{Outcome}(F_i)$ iff $\exists (q, a, \varphi, \lambda, q') \in E$ and one of the following condition holds:
  1. either $a \in Act_i$ and $\exists Z''. F_i((q, Z'')) = a$ and $(q', Z') = \text{Post}_a((q, Z \cap Z'')) \nearrow^{F_i}$,
  2. or $a \in Act_o$ and $\exists Z''. F_i((q, Z'')) = \text{delay}$ and $(q', Z') = \text{Post}_a((q, Z \cap Z'')) \nearrow^{F_i}$.

The backward exploration ends when the set of winning states $PWin[X_0]$ contains the initial state. Then, the projection $(PWin[X_0] \cap (q_0, \mathbf{0}))_{|\Delta}$ 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.

Algorithm 2: Counter strategy refinement.

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

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

1 begin

$\quad$ /* Initialization */

2 $\quad$ $Waiting \leftarrow \emptyset$

3 $\quad$ **for** $X = (q, Z) \in \mathcal{Z}_{\mathcal{A}}^{\Delta_{bad}}$ **do**

4 $\quad\quad$ **if** $q \in \mathsf{Bad}$ **then**

5 $\quad\quad\quad$ $PWin[X] \leftarrow (q, [\![Inv(q)]\!])$

6 $\quad\quad\quad$ $Waiting \leftarrow Waiting \cup \{Y \mid \exists \rho.\ \rho \rightarrow Y \rightarrow X \in \mathsf{Outcome}(F_i)\}$

7 $\quad\quad$ **else**

8 $\quad\quad\quad$ $PWin[X] \leftarrow \emptyset$

9 $\quad\quad$ **end**

10 $\quad$ **end**

$\quad$ /* Backward exploration */

11 $\quad$ **while** $(Waiting \neq \emptyset) \wedge (q_0, \mathbf{0}) \notin PWin[X_0]$ **do**

12 $\quad\quad$ $X = (q, Z) \leftarrow \mathsf{pop}(Waiting)$

13 $\quad\quad$ $PBad^* \leftarrow (q, \neg [\![Inv(q)]\!]) \cup (\bigcup_{X \xrightarrow{a \in Act_i} Y} \mathsf{PPred}_a(Win[Y]))$

14 $\quad\quad$ $PGood^* \leftarrow \bigcup_{X \xrightarrow{a \in Act_o} Y} \mathsf{PPred}_a((q, [\![Inv(q)]\!]) \setminus PWin[Y])$

15 $\quad\quad$ $PWin[X] \leftarrow \mathsf{PPred}_t(PBad^*, PGood^* \setminus PBad^*)$

16 $\quad\quad$ $Waiting \leftarrow Waiting \cup \{Y \mid \exists \rho.\ \rho \rightarrow Y \rightarrow X \in \mathsf{Outcome}(F_i)\}$

17 $\quad$ **end**

18 $\quad$ **return** $\mathsf{Inf}((PWin[X_0] \cap (q_0, \mathbf{0}))_{|\Delta})$

19 end

Theorem 7 *(Soundness). Algorithm 2 returns a value $\Delta_{inf} \in \mathbb{Q}$ such that $\forall \Delta > \Delta_{inf}$ the game $(\mathcal{A}_{\mathrm{rob}}^{\Delta}, W^{\mathsf{o}}(\mathsf{Bad}))$ is lost.*

Proof. Given a value $\Delta \in \mathbb{Q}$, Algorithm 2 is similar to the timed games algorithm of TIGA described in [28], but with less exploration steps since only the outcome of the spoiling strategy are explored. Therefore for that value $PWin[X_0] \subseteq Win[X_0]$.

By assumption, the game $(\mathcal{A}_{\mathrm{rob}}^{\Delta_{bad}}, W^{\mathsf{o}}(\mathsf{Bad}))$ is lost, so $\Delta_{bad} \in (PWin[X_0] \cap (q_0, \mathbf{0}))_{|\Delta}$.

Ab absurdo, if $\Delta > \Delta_{inf}$ is a good value, *i.e.* $(\mathcal{A}_{\mathrm{rob}}^{\Delta}, W^{\mathsf{o}}(\mathsf{Bad}))$ is winning, then $(q_0, \mathbf{0}) \notin Win[X_0]$, and consequently $(q_0, \mathbf{0}) \notin PWin[X_0]$. This implies that $(PWin[X_0] \cap (q_0, \mathbf{0}))_{|\Delta} = \emptyset$, which is a contradiction. $\quad\square$

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

7. Implementation and experiments

7.1. PyECDAR implementation

The specification theory described in [13] is implemented in the tool ECDAR [37]. In order to experiment with the methods proposed in the present paper, we have built a prototype tool in Python that reimplements the main functionalities of ECDAR and supports the analyses of the robustness of timed specifications [38]. 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 $\Delta \in \mathbb{Q}_{\geqslant 0}$.
2. Computation of the robust game automaton $\mathcal{A}_{\mathrm{rob}}^{\Delta}$.
3. In order to add rational perturbations on the models $\mathcal{I}_{\Delta}$ and $\mathcal{A}_{\mathrm{rob}}^{\Delta}$ 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 [28] and the on-the-fly timed game algorithm presented in [30].

Table 1
Robust consistency of the university specifications.

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

Table 2
Robust compatibility between the university specifications.

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

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 Algorithm 2 operations on parametric symbolic states are handled with the Parma Polyhedra Library [39]. 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 is not such 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 [35].

7.2. Experiments

We evaluate the performance of the tool to solve the parametric robustness problems on two 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 [13]. They are part of an overall specification of a university, composed by three specifications: the coffee machine (M) of Fig. 1(a), the researcher (R) of Fig. 1(b), and the administration (A) (see [13]). 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 edges. 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 not sensitive to the values of the two initial parameters Δ_{init} and ϵ. This is not the case for Binary Search: the precision ϵ determines 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.

Specification of a Milner scheduler. The second experiment studies a real-time version of Milner's scheduler previously introduced in [37]. 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 not sensitive to the initial conditions and in general more efficient than Binary Search.

7.3. Interpretation

Previous results are summarized in Fig. 11 in order to compare the performance of the two methods on the most complex examples.

The performance of the Binary Search method depends on the number of games that are solved and on the outcome of these games. Games that are winning (or games that are losing but with a value of Δ close to the optimum value) are harder to solve, since in these cases the (almost) complete symbolic state space must be explored. Reducing the precision parameter ϵ implies that more games must be solved close to the optimum value, and therefore it increases the time of analysis. Moreover, changing, even slightly, the initial maximum value Δ_{init} may change the number of games, but most

Table 3
Robust consistency of Milner's scheduler nodes.

Model	Game size		$\Delta_{init} = 30$ $\epsilon = 0.5$		$\Delta_{init} = 31$ $\epsilon = 0.5$		$\Delta_{init} = 30$ $\epsilon = 0.1$		$\Delta_{init} = 31$ $\epsilon = 0.1$	
	loc.	edges	CR	BS	CR	BS	CR	BS	CR	BS
1 Node	13	35	0.97 s	0.68 s	1.09 s	0.72s	0.97 s	1.03 s	1.09 s	1.09s
2 Node s	81	344	10.7 s	10.3 s	11.2 s	12.6 s	10.5 s	15.8 s	11.1 s	19.4 s
3 Node s	449	2640	1m58	2m25	2m06	2m26	1m57	3m39	2m05	3m45
4 Node s	2305	17 152	17m38	24m12	17m38	27m46	17m41	37m57	17m37	41m50

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

(b) Milner experiment
4 nodes model
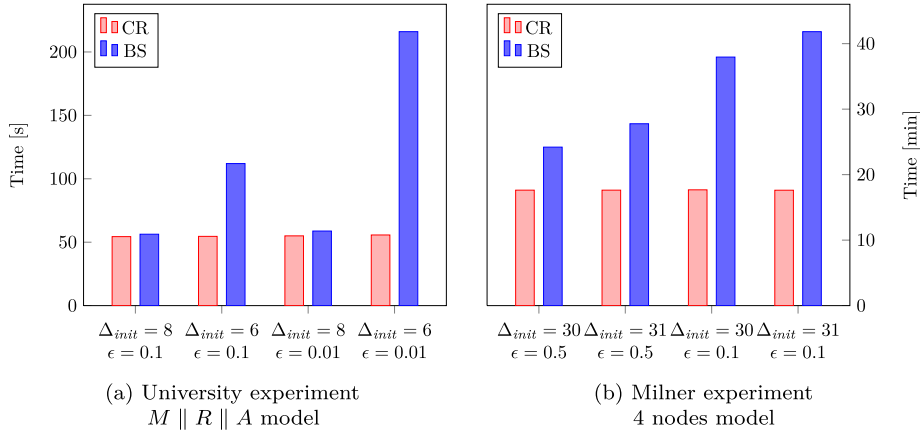
Fig. 11. Comparisons of the performances between the two methods Counter strategy Refinement (CR) and Binary Search (BS).

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 solved 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 solves the game with the value $\Delta_{min} - \epsilon$. The experiments shows this has no impact on the performances.

7.4. Parking controller

In this final case study we analyze the robustness of a parking controller. This component is part of parking system, described in [40], that model the behavior of a car park. The system is composed by an entry gate, an exit gate, a gate controller and a payment machine. We study the implementation of the gate controller whose task is to deliver entry tickets, whenever they are requested, and as long as the parking is not full.

The specification Controller of this components is given in Fig. 12 for a capacity N_{max} of cars in the parking. It specifies that tickets must be delivered at most 10 time units after accepting a request and the request is accepted only if the parking is not full. It also specifies some assumptions about the environment by linking the unexpected behaviors to a universal location l_u. These assumptions are that a vehicle may exit only when the parking is not empty, that it may enter only after receiving its entry tickets, and that tickets may be requested at least 6 time units after the previous one has been delivered.

We study the robust consistency of this specification in order to determine if we can implement it. The results are given in Table 4. For an increasing number N_{max} of vehicles we list the maximum allowed perturbation Δ_{max}, and the time needed to compute it with two values of precision for each methods. The results show that the robustness of the specification decreases when the capacity of the parking increases. We also remark that the counterexample refinement method is less efficient than binary search to analyze this model. Indeed on this model the counterexample refinement method requires more games to determine that value of Δ_{max}.

This low robustness can be a problem to implement the controller for large number of vehicles. We can analyze spoiling strategies to find that the responsible executions involve a large number of vehicle exiting the parking within the short time interval before the entry ticket is issued. However this is unrealistic and therefore we propose to fix the model. We add an assumption on the environment that limits the number of vehicles that can exit within a given time interval. This assumption is specified in a new specification Assumption shown in Fig. 13. It allows any behaviors, but, when a
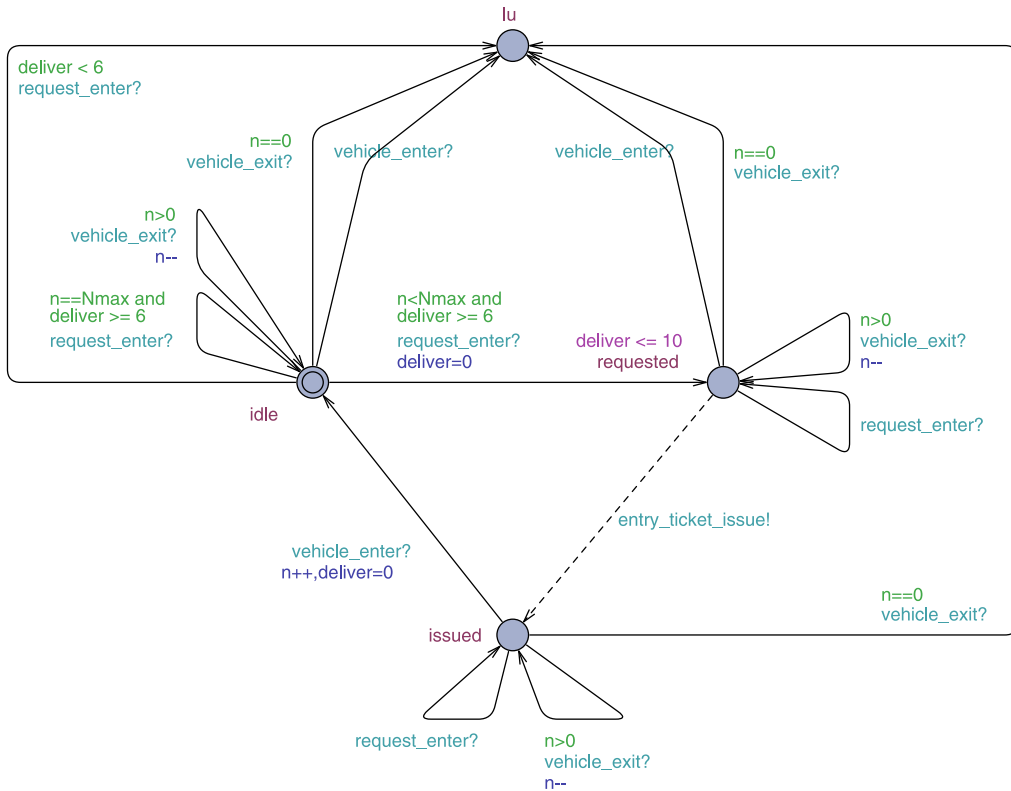
Fig. 12. Specification of the parking controller.

Table 4
Robust consistency of the parking controller.

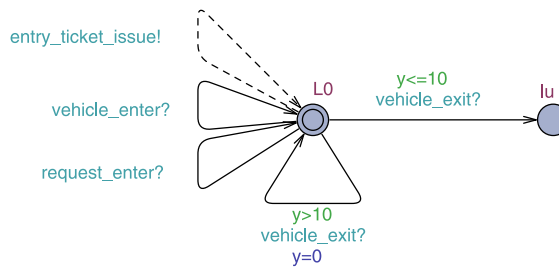| Number of vehicles N_{max}	Δ_{max}	$\epsilon = 0.1$		$\epsilon = 0.01$	
		CR	BS	CR	BS
1	5	78 ms	204 ms	78 ms	288 ms
2	2.5	376 ms	547 ms	367 ms	788 ms
5	1	2.82 s	1.71 s	2.82 s	2.58 s
10	0.5	7.05 s	3.78 s	13.3 s	6.29 s
15	0.33	12.9 s	5.62 s	35.6 s	10.9 s
20	0.25	14.74 s	10 s	59.44 s	19.1 s

Fig. 13. Assumption on the environment of the parking controller.

vehicle exit less than 10 time units before the previous one, it goes to the universal location. We add this assumption in conjunction with the controller specification. We analyze the robustness of the new specification Controller ∧ Assumption, and, as previously, we list the results in Table 5. It shows that the robustness now remains constant for any number of vehicles.

Table 5

Robust consistency of the parking's controller with assumption.

Number of vehicles	Δ_{max}	$\epsilon = 0.1$		$\epsilon = 0.01$	
		CR	BS	CR	BS
1	5	205 ms	619 ms	205 ms	883 ms
2	2.5	1.11 s	1.7 s	1.11 s	2.52 s
5	2.5	2.42 s	3.26 s	2.42 s	4.34 s
10	2.5	5.6 s	6.53 s	5.61 s	7.69 s
15	2.5	8.91 s	9.86 s	8.93 s	11.2s
20	2.5	12.7 s	14 s	12.7 s	15.5 s

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 [13] 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 [16] to the setting of specification theories, to solve robust games by reducing them to problems on classical timed games.

As a new contribution from [27], we also study the parametric robustness problems and evaluate the maximum imprecision allowed by specifications. To this end we propose a counterexample 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 counterexample refinement approach on the alternating simulation game, in order to solve the parametric satisfaction problem for an existing implementation.

In future we plan to extend our approach to different models, like timed automata with stochastic semantics [41]. 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, http://www.omg.org/spec/CORBA/3.2/, 2011.
[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, http://www.w3.org/TR/wsdl20/, 2007.
[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. Exp. 41 (2011) 133–142.
[8] A. Cimatti, Industrial applications of model checking, in: Modeling and Verification of Parallel Processes, in: Lect. Notes Comput. Sci., vol. 2067, Springer, 2001, pp. 153–168.
[9] B. Long, J. Dingel, T. Graham, Experience applying the spin model checker to an industrial telecommunications system, in: ICSE '08, pp. 693–702.
[10] D. Keating, A. McInnes, M. Hayes, An industrial application of model checking to a vessel control system, in: Electronic Design, Test and Application (DELTA), 2011, pp. 83–88.
[11] F. Kammüller, S. Preibusch, An industrial application of symbolic model checking, Inform. Forsch. Entwickl. 22 (2008) 95–108.
[12] S. Chandra, P. Godefroid, C. Palm, Software model checking in practice: an industrial case study, in: ICSE '02, pp. 431–441.
[13] 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.
[14] L. de Alfaro, T.A. Henzinger, Interface automata, in: ESEC/SIGSOFT FSE, pp. 109–120.
[15] L. de Alfaro, T.A. Henzinger, M. Stoelinga, Timed interfaces, in: EMSOFT, in: Lect. Notes Comput. Sci., vol. 2491, Springer, 2002, pp. 108–122.
[16] K. Chatterjee, T.A. Henzinger, V.S. Prabhu, Timed parity games: Complexity and robustness, in: FORMATS, in: Lect. Notes Comput. Sci., vol. 5215, Springer, Saint Malo, France, 2008, pp. 124–140.
[17] The COMBEST Consortium, Combest, http://www.combest.eu, 2008–2011.
[18] The SPEEDS Consortium, Speeds, http://www.speeds.eu.com, 2006–2010.
[19] 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.
[20] A. Puri, Dynamical properties of timed automata, in: Formal Techniques in Real-Time and Fault-Tolerant Systems, in: Lect. Notes Comput. Sci., vol. 1486, Springer, 1998, pp. 210–227.
[21] M.D. Wulf, L. Doyen, J.-F. Raskin, Almost ASAP semantics: from timed models to timed implementations, Form. Asp. Comput. 17 (2005) 319–341.
[22] M. Wulf, L. Doyen, N. Markey, J.-F. Raskin, Robust safety of timed automata, Form. Methods Syst. Des. 33 (2008) 45–84.
[23] P. Bouyer, K.G. Larsen, N. Markey, O. Sankur, C. Thrane, Timed automata can always be made implementable, in: CONCUR, Aachen, Germany, in: Lect. Notes Comput. Sci., vol. 6901, Springer, 2011, pp. 76–91.
[24] O. Sankur, P. Bouyer, N. Markey, Shrinking timed automata, in: FSTTCS, Mumbai, India, in: Leibniz International Proceedings in Informatics, Leibniz-Zentrum für Informatik, 2011, pp. 90–102.
[25] P. Bouyer, N. Markey, O. Sankur, Robust model-checking of timed automata via pumping in channel machines, in: FORMATS, Aalborg, Denmark, in: Lect. Notes Comput. Sci., vol. 6919, Springer, 2011, pp. 97–112.

[26] R. Jaubert, P.-A. Reynier, Quantitative robustness analysis of flat timed automata, in: FOSSACS, in: Lect. Notes Comput. Sci., vol. 6604, Springer, 2011, pp. 229–244.

[27] K.G. Larsen, A. Legay, L.-M. Traonouez, A. Wasowski, Robust specification of real time components, in: FORMATS, Aalborg, Denmark, in: Lect. Notes Comput. Sci., vol. 6919, Springer, 2011, pp. 129–144.

[28] F. Cassez, A. David, E. Fleury, K.G. Larsen, D. Lime, Efficient on-the-fly algorithms for the analysis of timed games, in: CONCUR, in: Lect. Notes Comput. Sci., vol. 3653, Springer, 2005, pp. 66–80.

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

[30] P. Bulychev, T. Chatain, A. David, K.G. Larsen, Efficient on-the-fly algorithm for checking alternating timed simulation, in: FORMATS, in: Lect. Notes Comput. Sci., vol. 5813, Springer, 2009, pp. 73–87.

[31] 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, in: Lect. Notes Comput. Sci., vol. 2761, Springer, 2003, pp. 142–156.

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

[33] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement, in: CAV, in: Lect. Notes Comput. Sci., vol. 1855, Springer, 2000, pp. 154–169.

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

[35] 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.

[36] G. Behrmann, A. Cougnard, A. David, E. Fleury, K.G. Larsen, D. Lime, Uppaal-tiga: Time for playing games!, in: CAV, in: Lect. Notes Comput. Sci., vol. 4590, Springer, 2007, pp. 121–125.

[37] 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, in: Lect. Notes Comput. Sci., vol. 6252, Springer, Singapore, 2010, pp. 365–370.

[38] Python implementation of ECDAR, Pyecdar, https://project.inria.fr/pyecdar, 2011.

[39] 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, Sci. Comput. Program. 72 (2008) 3–21.

[40] J.-B. Raclet, B. Caillaud, D. Nickovic, R. Passerone, A. Sangiovanni-Vincentelli, T. Henzinger, K.G. Larsen, Contracts for the design of embedded systems, Part I: Methodology and use cases, Technical Report, 2012, submitted for publication, http://www.irisa.fr/distribcom/benveniste/pub/ProcIEEE_contractsPart1.pdf.

[41] 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, Aalborg, Denmark, in: Lect. Notes Comput. Sci., vol. 6919, Springer, 2011, pp. 80–96.