Robust Specification of Real Time Components

Kim G. Larsen,[1] Axel Legay,[2] Louis-Marie Traonouez,[3] Andrzej Wąsowski[3]

[1] Aalborg University, Denmark, kgl@cs.aau.dk
[2] INRIA Rennes, France, axel.legay@irisa.fr
[3] IT University of Copenhagen, Denmark, {lmtr,wasowski}@itu.dk

Abstract. Specification theories for real-time systems allow to reason 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 be able to reason about possibility to effectively implement the theoretical specifications on physical systems. In the literature, this implementation problem has already been linked to the robustness problem for Timed Automata, where small perturbations in the timings of the models are introduced. We address the problem of robust implementations in timed specification theories. Our contributions include the analysis of robust timed games and the study of robustness with respect to the operators of the theory.

1 Introduction

For long, software engineers have practiced component-oriented software construction, building systems from modules that only depend on each other in well specified ways. Foundational research follows up by developing trustworthy rigorous methods for component-oriented design. In concurrency theory 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 and conjunction (logical composition).

Specification theories integrate prior results to provide a uniform design method. In [1], we have proposed the first complete specification theory for timed systems. We build on an input/output extension of the classical timed automata model—inputs are used to represent behaviors of the environment and outputs represent the behavior of the component. The theory is equipped with a game-based semantic, which is used to define all the good operations for such specification theory, including satisfaction (can a specification be implemented), refinement (how to compare specifications), logical composition (computing the intersection of two sets of implementations), structural composition (building large components from smaller ones), and quotient (synthesizing a component in a large design).

The theory in [1] is equipped with a consistency check that allows to decide whether a specification can indeed be implemented. Unfortunately, this check does not take imprecision of the physical world into account, that is consistency can be used to synthesize an implementation that may be not robust with respect to variations of

the environment. In practice, one would want to guarantee that a perturbation of the implementation still matches the requirements of the specification. Providing a solution to this problem in the setting of timed I/O specifications is our objective in this paper.

Our contributions include:

- We propose a notion of implementation of a specification that is robust with respect to a given perturbation in the delay before an action. Such perturbation is fixed in advance, which is a reasonable assumption as sensitivity with respect to perturbation of the environment is generally given in the description of the component that is provided by the manufacturer. The concept of robust implementation is lifted to a robust satisfaction operation that takes variations of timed behaviors into account when checking whether the implementation matches the requirement of the specification.
- 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 [2], Chatterjee et al. showed that problems on robust timed games can be reduced to classical problems on an extended timed game. We modify the original construction of [2] 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, classical compositional design operators are lifted to the robust setting. One of the nice features of our approach is that this does not require to modify the definitions of the operators themselves and that all the good properties of a specification theory (including independent implementability) are maintained.

To the best of our knowledge, this paper presents the first complete theory for robust timed specification. While the presentation is restricted to the theory of [1], we believe that the approach works for any timed specifications. Our experience with industrial projects shows that such realistic design theories are of clear interest [3,4,5].

State of The Art. None of the existing specification theories for timed systems allows for the treatment of robustness.

Various works have considered robustness 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 [6]. The problem has been linked to the implementation problem in [7], 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 [8] that proposes another robust semantics with both clock drifts and guard enlargement. Extending [6] they solve the robust safety problem, defined as the existence of a non-null value for the delays. 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 [7,8] 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 approach largely inspired by Chatterjee et al. [2], 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 [2] by allowing negative perturbation of delays.

We proceed by introducing the background on Timed Specifications (Section 2). In Section 3 we introduce methods for solving robust time games that arise in our specification theory. These methods are used in Sections 4–5 to reason about consistency, conjunction, parallel composition and synthesis of specifications and robust implementations of real time components.

2 Background on Timed I/O Specifications

We now recall the definition of Timed I/O specifications [1]. We use $\mathbb{Z}$ (respectively $\mathbb{N}$) for the set of all integer numbers (resp. non-negative integers), $\mathbb{R}$ for the set of all real numbers, and $\mathbb{R}_{\geq 0}$ (resp. $\mathbb{R}_{>0}$) for the non-negative (resp. strictly positive) subset of $\mathbb{R}$. Rational numbers are denoted by $\mathbb{Q}$, and their subsets are denoted analogously. For $x \in \mathbb{R}_{\geq 0}$, let $\lfloor x \rfloor$ denote the integer part of x and $\langle x \rangle$ denote its fractional part.

In the framework of [1], specifications and their implementations are semantically represented by Timed I/O Transition Systems (TIOTS) that are nothing more than timed transition systems with input and output modalities on transitions. Later we shall see that input represents the behaviors of the environment in which a specification is used, while output represents behaviours of the component itself.

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

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

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

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

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

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

a valuation u and $d \in \mathbb{R}_{\geq 0}$, we write $u+d$ for the valuation in which for each clock $x \in X$ we have $(u+d)(x) = u(x)+d$. For $Y \subseteq X$, we write $u[Y \mapsto 0]$ for a valuation agreeing with u on clocks in $X \setminus Y$, and giving 0 for clocks in Y.

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

Definition 2. *A* Timed I/O Automaton *is a tuple $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$, where Loc is a finite set of* locations*, $q_0 \in Loc$ is the* initial location*, Clk is a finite set of* clocks*, $E \subseteq Loc \times Act \times \mathcal{C}(Clk) \times 2^{Clk} \times Loc$ is a set of* edges*, $Act = Act_{\mathrm{i}} \oplus Act_{\mathrm{o}}$ is a finite set of* actions*, partitioned into* inputs *(Act_{i}) and* outputs *(Act_{o}), $Inv : Loc \mapsto \mathcal{C}(Clk)$ is a set of location* invariants*. Without loss of generality we assume that invariants of a location are always included in the guards of the edges that are incident with the location. We also assume that guards are satisfiable (for any guard φ the set $[\![\varphi]\!]$ is non-empty).*

A *universal location*, denoted l_{u}, in a TIOA accepts every input and can produce every output at any time. Location l_{u} models an unpredictable behavior of a component.

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

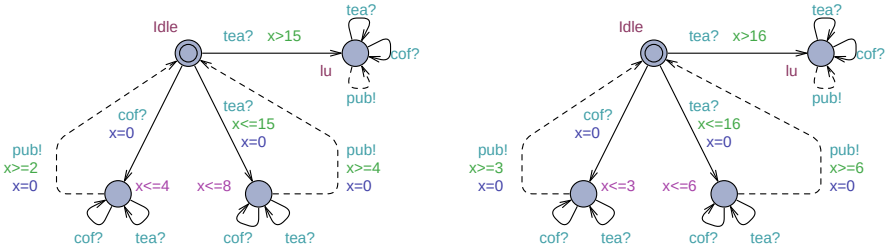
- Each edge $(q, a, \varphi, \lambda, q') \in E$ gives rise to $(q, u) \xrightarrow{a} (q', u')$ for each clock valuation $u \in \mathbb{R}^{Clk}_{\geq 0}$ such that $u \models \varphi$ and $u' = u[\lambda \mapsto 0]$ and $u' \models Inv(q')$.
- Each location $q \in Loc$ with a valuation $u \in \mathbb{R}^{Clk}_{\geq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

Since TIOTSs are infinite size they cannot be directly manipulated by computations. Usually *symbolic representations*, such as *region graphs* [9] or *zone graphs*, are used as data structures that finitely represent semantics of TIOAs. Let M be the greatest (in absolute value) integer constant that appears in the guards of a TIOA [4]. 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 $v(x) > M$,
- $\forall x, y \in Clk$, $\forall c \in [-M, M]$, $u(x) - u(y) \leq c$ iff $v(x) - v(y) \leq c$,
- $\forall x \in Clk$ if $u(x) \leq M$ then $\langle u(x) \rangle = 0$ iff $\langle v(x) \rangle = 0$,

We write $r\nearrow$ for the direct time successor of the region r, if such exists. The *region graph* of a TIOA $\mathcal{A}$ is $\mathcal{G} = (\mathcal{R}_{\mathcal{A}}, \rightarrow^G)$, where $\mathcal{R}_{\mathcal{A}} = \{(q, r) \mid \exists (q, u) \in St^{[\![\mathcal{A}]\!]_{\mathrm{sem}}}. u \in r\}$ is the set of regions, and $\rightarrow^G \subseteq \mathcal{R}_{\mathcal{A}} \times (Act \cup \{\tau\}) \times \mathcal{R}_{\mathcal{A}}$, such that $(q, r) \xrightarrow{\tau}{}^G (q, r\nearrow)$ iff $r\nearrow \models Inv(q)$, and $(q, r) \xrightarrow{a}{}^G (q', r')$ iff $(q, u) \xrightarrow{a} (q', u')$ for some $u \in r$ and $u' \in r'$.

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



(a) A specification (b) An implementation

Fig. 1: TIOAs of a specification and an implementation for a researcher.

Basics of the Timed Specification Theory. In [1], timed specifications and implementations are both represented by TIOAs satisfying additional conditions:

Definition 3. *A* specification $\mathcal{S}$ *is a TIOA whose semantics* $[\![\mathcal{S}]\!]_{\text{sem}}$ *is deterministic and input-enabled. An* implementation $\mathcal{I}$ *is a specification whose semantics* $[\![\mathcal{I}]\!]_{\text{sem}}$ *additionally verifies the output urgency and the independent progress conditions.*

Example 1. Figure 1a presents a specification of a researcher. It accepts either coffee (coff) 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 a universal state l_{u}. An individual researcher is an implementation of this specification. One example is presented in Figure 1b.

In specification theories, a *refinement* relation plays a central role. It allows to compare specifications, and to relate implementations to specifications. In [1], as well as in [10,11,12], refinement is defined in the style of alternating (timed) simulation:

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

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

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

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

The reader might find it surprising that in a robust specification theory we refrain from adjusting the refinement to account for imprecision of implementations when comparing specifications. Our basic assumption is that specifications are precise mathematical

objects that are not susceptible to imprecision of execution. In contrary, implementations can behave imprecisely when executed, so in Section 3 we will introduce an extension of Def. 5 that takes this into account. It is a fortunate property of Def. 4 that we do not need to modify it in order to reason about robust implementations (Property 3 in Sect. 3).

In [1], 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 [1], which will be illustrated in Sect. 4–5.

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

Definition 6. *A memoryless* strategy f *for the output player on the TIOA* $\mathcal{A}$ *is a function* $St^{[\![\mathcal{A}]\!]_{\mathrm{sem}}} \mapsto Act_{\mathrm{o}} \cup \{\mathsf{delay}\}$*, such that whenever* $f(s) \in Act_{\mathrm{o}}$ *then* $s \xrightarrow{f(s)} s'$ *for some* s'*, and whenever* $f(s) = \mathsf{delay}$ *then* $s \xrightarrow{d} s''$ *for some* $d > 0$ *and state* s''*.*

The restricted behavior of the TIOA when one player applies a specific strategy is defined as the *outcome* of the strategy.

Definition 7. *Let* $\mathcal{A}$ *be a TIOA,* f *a strategy over* $\mathcal{A}$ *for the output player, and* s *a state of* $[\![\mathcal{A}]\!]_{\mathrm{sem}}$*. The outcome* $\mathsf{Outcome}_{\mathrm{o}}(s, f)$ *of* f *from* s *is the subset of* $\mathsf{Runs}(s, [\![\mathcal{A}]\!]_{\mathrm{sem}})$ *defined inductively by:*

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

A *winning condition* for a player in the TIOA $\mathcal{A}$ is a subset of $\mathsf{Runs}([\![\mathcal{A}]\!]_{\mathrm{sem}})$. In safety games the winning condition is to avoid a set Bad of "bad" states (without lost of generality we assume these "bad" states correspond to a set of entirely "bad" locations). Formally, the winning condition is $WS^{\mathrm{o}}(\mathsf{Bad}) = \{\rho \in \mathsf{Runs}([\![\mathcal{A}]\!]_{\mathrm{sem}}) \mid \mathsf{States}(\rho) \cap \mathsf{Bad} = \emptyset\}$. A strategy f for output is *winning* from state s if $\mathsf{Outcome}_{\mathrm{o}}(s, f) \subseteq WS^{\mathrm{o}}(\mathsf{Bad})$. A state s is winning if there exists a winning strategy from s. The game $(\mathcal{A}, WS^{\mathrm{o}}(\mathsf{Bad}))$ is winning if the initial state is winning. Solving this game is decidable [13,14,1].

Strategies can also be defined symbolically, using the region graph introduced in the previous section. For a region (q, r), if $f(q, r) = \mathsf{delay}$ then $(q, r) \xrightarrow{\tau}{}^{G} (q, r\nearrow)$, and if $f(q, r) \in Act_{\mathrm{o}}$ then $\exists (q', r').(q, r) \xrightarrow{a}{}^{G} (q', r')$. An outcome of f is then a run in the region graph, such that if $\rho \in \mathsf{Outcome}_o((q, r), f)$ then $\rho' = \rho \xrightarrow{a} (q', r') \in \mathsf{Outcome}_o((q, r), f)$ if $\mathsf{last}(\rho) \xrightarrow{a}{}^{G} (q', r')$, and either $a \in Act_{\mathrm{i}}$, or $a \in Act_{\mathrm{o}}$ and $f(\mathsf{last}(\rho)) = a$, or $a = \tau$ and $f(\mathsf{last}(\rho)) = \mathsf{delay}$.

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

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

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

It follows that no immediate error states can occur in implementations, since they verify independent progress. In [1] we show that $\mathcal{S}$ is consistent iff there exists a winning strategy for output in the safety game $(\mathcal{S}, WS^{\text{o}}(\mathsf{err}^{\mathcal{S}}))$. Moreover, the maximum consistent part of $\mathcal{S}$ corresponds to the maximum wining strategy for output in this game.

Conjunction of two specifications is found as a maximal strategy for output in a safety game on the product state space to avoid immediate errors. Similarly, optimistic parallel composition of two specifications is computed as a maximum strategy for input in a safety game over the product state space; and a quotient of two specifications is found as a maximum strategy for output in another safety game. Optimistic composition means that two specifications are compatible if there exists at least one environment, in which they can avoid error states. Details can be found in [1].

3 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 [7] in the context of model checking. In specification theories the small deficiencies of hardware can be reflected in a strengthened satisfaction relation, which introduces small perturbations to the timing of implementation actions, before they are checked against the requirements of a specification—ensuring that the implementation satisfies the specification even if its behavior is perturbed.

We first formalize the concept of perturbation. Let $\varphi \in \mathcal{C}(X)$ be a guard over the set of clocks X. The *enlarged guard* $\lceil \varphi \rceil_{\Delta}$ is constructed according to the following rules:

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

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

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

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

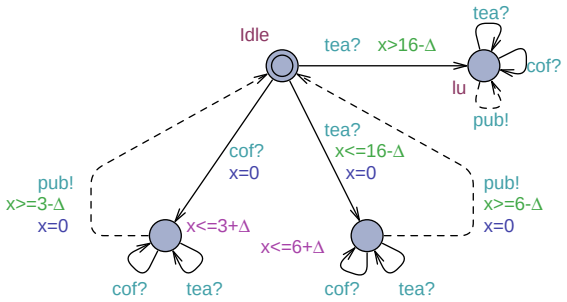


Fig. 2: Δ-perturbation of the researcher implementation.

Perturbed Implementation and Robust Timed I/O Specifications. We lift the perturbation to implementation TIOAs. Given a jitter Δ, the perturbation means a Δ-enlargement of invariants and of output edge guards. Guards on the input edges are restricted by Δ:

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

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

$\mathcal{I}_\Delta$ is not necessarily action deterministic, as output guards are enlarged. However it is input-enabled, since by construction (the last case above), any input not accepted after restricting input guards is redirected to the universal location l_u. Also $\mathcal{I}_0$ equals $\mathcal{I}$.

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

Such considerations are out of place in classical robustness theories for model checking, but are crucial when moving to models, where input and output transitions are distinguished. For example, in [7] 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 [7] both input and output guards are increased, which is suitable for the one-player setting, but incompatible with the contravariant nature of two-player games. Such enlargement would not be monotonic with respect to the alternating refinement (Def. 4), while the perturbation of Def. 8 is monotonic.

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

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

Property 1. Let $\mathcal{I}$ be an implementation and $0 \leq \Delta_1 < \Delta_2$. Then $\mathcal{I} \leq \mathcal{I}_{\Delta_1} \leq \mathcal{I}_{\Delta_2}$.

In addition, we obtain the following by transitivity of the refinement:

Property 2. Let $\mathcal{S}$ be a specification and $\Delta_1 \leq \Delta_2$, then $[\![\mathcal{S}]\!]^{\Delta_2}_{\text{mod}} \subseteq [\![\mathcal{S}]\!]^{\Delta_1}_{\text{mod}} \subseteq [\![\mathcal{S}]\!]_{\text{mod}}$.

Property 3. Let $\mathcal{S}$ and $\mathcal{T}$ be specifications and $0 \leq \Delta$, then $\mathcal{S} \leq \mathcal{T} \implies [\![\mathcal{S}]\!]^\Delta_{\text{mod}} \subseteq [\![\mathcal{T}]\!]^\Delta_{\text{mod}}$.

We now turn to the problem of deciding whether a specification is robustly consistent:

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

Like in the non-robust case, deciding consistency and performing operations on specifications are 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 Sections 4 and 5, we show how they can be used to perform classical operations on specifications.

Example 2. Figure 2 presents the Δ-perturbation of the researcher implementation presented in Fig. 1b. One can check that this implementation robustly satisfies the specification of Fig. 1a for any $\Delta \in]0, 1]$.

Robust Timed Games for Timed I/O Specifications. We first define robust strategies that guarantee winning even if subject to bounded timing perturbations. We then propose a technique for finding such strategies. We start with the construction of *syntactic outcome* that represents game outcomes as TIOAs. We rely on the region graph construction in the definition of syntactic outcome, but any stable partitioning of the state-space could serve this purpose, and would be more efficient in practice.

Definition 11. *Let $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$ be a TIOA and f a strategy over $\mathcal{A}$ for output. The TIOA $\mathcal{A}_f = (\mathcal{R}_\mathcal{A}, (q_0, r_0), Clk \cup \{z\}, \widehat{E}, Act \cup \{\tau\}, \widehat{Inv})$ is built by decorating the region graph $\mathcal{G} = (\mathcal{R}_\mathcal{A}, \rightarrow^G)$ of $\mathcal{A}$, using the original clocks of $\mathcal{A}$ and an additional clock z to impose output urgency. For each region (q, r), the incident edges and the invariant are defined as follows:*

- *$\widehat{Inv}(q, r) = Inv(q) \wedge (r \vee r\nearrow)$;*
- *If $(q, r) \xrightarrow{\tau}{}^G (q, r\nearrow)$ then $((q, r), \tau, r\nearrow, \{z\}, (q, r\nearrow)) \in \widehat{E}$;*
- *For each edge $(q, i?, \varphi, \lambda, q') \in E$, if $(q, r) \xrightarrow{i?}{}^G (q', r')$ then $((q, r), i?, \varphi, \lambda \cup \{z\}, (q', r')) \in \widehat{E}$;*
- *If $f(q, r) = o!$, then $\widehat{Inv}(q, r) = r \wedge \{z = 0\}$, and for each edge $(q, o!, \varphi, \lambda, q') \in E$, if $(q, r) \xrightarrow{o!}{}^G (q', r')$, then $((q, r), o!, \varphi, \lambda \cup \{z\}, (q', r')) \in \widehat{E}$.*

This construction captures the semantic outcome of the game in the following sense:

Proposition 1. *Let $(\mathcal{A}, W)$ be a timed safety game and f be a strategy for output. Then a run ρ in the region graph $\mathcal{G}$ of $\mathcal{A}$ is in* $\mathsf{Outcome}_{\mathrm{o}}((q,r), f)$ *iff ρ is an untimed run of $\mathcal{A}_f$ starting from region (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. We write $\lceil \mathcal{A} \rceil^{\mathrm{o}}_{\Delta}$ (resp. $\lceil \mathcal{A} \rceil^{\mathrm{i}}_{\Delta}$) for the TIOA where the guards of the output (resp. input) player and the invariants have been enlarged by Δ.

Definition 12. *For a timed game $(\mathcal{A}, W)$, a strategy f for output is* Δ-robust winning *if it is winning when the moves of output are perturbed, i.e.* $\mathsf{Runs}(\llbracket \lceil \mathcal{A}_f \rceil^{\mathrm{o}}_{\Delta} \rrbracket_{\mathrm{sem}}) \subseteq W$. [5]

As proposed in [2], robust timed games for a bounded delay can be reduced to classical timed games by a syntactic transformation of the game automaton. Here, we propose a modified version of the construction that respects the duality between inputs and outputs:

Definition 13. *Let $\mathcal{A} = (Loc, q_0, Clk, E, Act, Inv)$ be TIOA and a $\Delta \in \mathbb{Q}_{>0}$, the* robust game automaton $\mathcal{A}^{\Delta}_{\mathrm{rob}} = (\widetilde{Loc}, q_0, Clk \cup \{y\}, \widetilde{E}, \widetilde{Act}, \widetilde{Inv})$ *uses an additional clock y and is constructed according to the following rules:*

- *$Loc \subseteq \widetilde{Loc}$, and for each location $q \in Loc$ and each edge $e = (q, o!, \varphi, \lambda, q') \in E$, two locations q^{α}_e and q^{β}_e are added in $\widetilde{Loc}$. The invariant of q is unchanged; the invariants of q^{α}_e and q^{β}_e are $y \leq \Delta$.*
- *For each action $o! \in Act_{\mathrm{o}}$, an additional action $o?$ is added in $\widetilde{Act}_{\mathrm{i}}$.*
- *Each edge $e' = (q, i?, \varphi, \lambda, q') \in E$ gives rise to the following edges in $\widetilde{E}$: $(q, i?, \varphi, \lambda \cup \{y\}, q')$, $(q^{\alpha}_e, i?, \varphi, \lambda \cup \{y\}, q')$ and $(q^{\beta}_e, i?, \varphi, \lambda \cup \{y\}, q')$.*
- *Each edge $e = (q, o!, \varphi, \lambda, q') \in E$ gives rise to the following edges in $\widetilde{E}$: $(q, o!, \varphi\rfloor_{2\Delta}, \{y\}, q^{\alpha}_e)$, $(q^{\alpha}_e, o!, \{y = \Delta\}, \{y\}, q^{\beta}_e)$, $(q^{\alpha}_e, o?, \{y \leq \Delta\}, \lambda \cup \{y\}, q')$ and $(q^{\beta}_e, o?, \{y \leq \Delta\}, \lambda \cup \{y\}, q')$, where $\varphi\rfloor_{2\Delta}$ is constructed from φ by restricting upper bound constraints by 2Δ.*

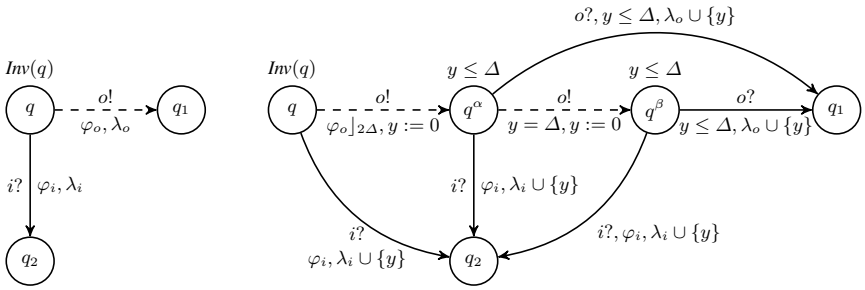
The construction is demonstrated in Fig 3. The output player can propose a move in the time interval $\lfloor \varphi \rfloor_{\Delta}$ (which is done in two steps: first playing $o!$ in the time interval $\varphi\rfloor_{2\Delta}$ and then a second firing after Δ time units), but the input player can perturb this move by choosing a smaller or greater delay to perform the action.

The construction shall serve as a tool for deciding robust consistency, synthesizing a robust implementation, and other operations of the specification theory with robustness.

Theorem 1. *For a timed safety game $(\mathcal{A}, W)$, if f is a winning strategy for output in the robust game $(\mathcal{A}^{\Delta}_{\mathrm{rob}}, W)$, then the following strategy f' is a Δ-robust winning strategy for output in the game $(\mathcal{A}, W)$: $\forall (q,r) \in \mathcal{R}_{\mathcal{A}}$, $f'(q,r) = o!$ if $\exists e. f(q^{\alpha}_e, \widetilde{r}) = o!$, where $\widetilde{r}$ is a region of $\mathcal{R}_{\mathcal{A}^{\Delta}_{\mathrm{rob}}}$, and r its projection on $\mathcal{R}_{\mathcal{A}}$ [6], otherwise $f'(q,r) =$* delay.

[5] Technically, we assume that runs in $\mathsf{Runs}(\llbracket \lceil \mathcal{A}_f \rceil^{\mathrm{o}}_{\Delta} \rrbracket_{\mathrm{sem}})$ abstract τ transitions

[6] To this end the region graph of $\mathcal{A}$ must be computed with Δ as the lowest constant.



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

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

Our method and our notion of robust strategy guarantees non-zenoness for the synthesized strategies. Indeed by allowing the opponent to perturb the verifier by Δ or $-\Delta$ we impose that the verifier only performs non-null delay actions. Later, in the context of timed specifications, this ensures realizability of implementations. Also non-Zeno environements are used as witnesses of compatibility in optimistic composition, which could have happened in [1], which ignored Zeno-problems altogether.

4 Robust Consistency and Conjunction

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 gives a method to synthesize a robust implementation of the specification.

Intuitively a specification is Δ-robust with respect to input $i?$, if between enabling of any two $i?$ edges at least 2Δ time passes, during which the reaction to $i?$ is unspecified. So, if the two transitions triggers Δ-too-late and Δ-too-early (respectively), there is no risk that the reaction is resolved non-deterministically in the specification.

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

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

For a specification $\mathcal{S}$, the safety objective for the robust consistency game is to avoid the set of states of error states $\mathsf{err}^{\mathcal{S}}_{\Delta}$ such that $(q, u) \in \mathsf{err}^{\mathcal{S}}_{\Delta}$ iff (q, u) violates independent progress or Δ-latency for inputs, so:

- Violates independent progress: $(\exists d \in \mathbb{R}_{\geq 0}.(q, u) \xnrightarrow{d})$ and $(\forall d \forall o!.(q, u) \xrightarrow{d} (q, u + d) \Rightarrow (q, u + d) \xnrightarrow{o!})$,

– Violates Δ-latency of inputs: $\exists e = (q, i?, \varphi, c, q')$, $q' \neq l_{\mathrm{u}}$, 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 $\mathsf{err}^{\mathcal{S}} \subseteq \mathsf{err}^{\mathcal{S}}_{\Delta}$, because the error condition with robustness is weaker than in the classical case (cf. page 7).

The robust safety game $(\mathcal{S}, \mathit{WS}^{\mathrm{o}}(\mathsf{err}^{\mathcal{S}}_{\Delta}))$ can be solved with the method presented in the previous section, and a winning strategy f for the game can be synthesized. Let $\mathcal{S}_f$ be the syntactic outcome of f in $\mathcal{S}$. We build from a 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 available in $\mathcal{S}_f$ might not be firable from $r \vee r\nearrow$. Then, in order to check the robust satisfaction relation between $(\mathcal{I}_f)_{\Delta}$ and $\mathcal{S}$, we add additional input edges to $\mathcal{I}_f$: for each location (q, r) in $\mathcal{S}_f$, for each edge $e = ((q, r), i?, \varphi, \lambda \cup \{z\}, (q^*, r^*))$ (with $q^* \neq l_{\mathrm{u}}$), and for each location (q', r') linked to (q, r) by a sequence of τ transitions, we add an edge $e' = ((q', r'), i?, \varphi, \lambda \cup \{z\}, (q^*, r^*))$.
– To support restriction of input guards in $(\mathcal{I}_f)_{\Delta}$, we replaced in $\mathcal{I}_f$ all guards φ of edges $e = ((q, r), i?, \varphi, \lambda \cup \{z\}, (q', r'))$ with $q' \neq l_{\mathrm{u}}$ by their elargement $\lceil \varphi \rceil_{\Delta}$. 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 and thus avoid multiplying input edges.

Theorem 2. *For a specification $\mathcal{S}$ and a robust winning strategy f in the Δ-robust consistency game, we have that $\mathcal{I}_f$ is a Δ-robust implementation of $\mathcal{S}$, i.e. $\mathcal{I}_f \ \mathbf{sat}_{\Delta}\ \mathcal{S}$.*

Conjunction. A conjunction of two specifications captures the intersection of their implementation sets. The following conjunction operator has been proposed in [1]:

Definition 14. *Let $\mathcal{S} = (\mathit{Loc}^S, q_0^S, \mathit{Clk}^S, E^S, \mathit{Act}, \mathit{Inv}^S)$ and $\mathcal{T} = (\mathit{Loc}^T, q_0^T, \mathit{Clk}^T, E^T, \mathit{Act}, \mathit{Inv}^T)$ be specifications that share the same alphabet of actions Act. We define their conjunction, denoted $\mathcal{S} \wedge \mathcal{T}$, as the TIOA $(\mathit{Loc}, q_0, \mathit{Clk}, E, \mathit{Act}, \mathit{Inv})$ where $\mathit{Loc} = \mathit{Loc}^S \times \mathit{Loc}^T$, $q_0 = (q_0^S, q_0^T)$, $\mathit{Clk} = \mathit{Clk}^S \uplus \mathit{Clk}^T$, $\mathit{Inv}((q_s, q_t)) = \mathit{Inv}(q_s) \wedge \mathit{Inv}(q_t)$, and the set of edges is defined by the following rule: if $(q_s, a, \varphi_s, c_s, q_s') \in E^S$ and $(q_t, a, \varphi_t, c_t, q_t') \in E^T$ then $((q_s, q_t), a, \varphi_s \wedge \varphi_t, c_s \cup c_t, (q_s', q_t')) \in E$.*

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

Theorem 3. *For specifications $\mathcal{S}$, $\mathcal{T}$ and $\Delta \in \mathbb{Q}_{>0}$: $[\![\mathcal{S} \wedge \mathcal{T}]\!]^{\Delta}_{\mathrm{mod}} = [\![\mathcal{S}]\!]^{\Delta}_{\mathrm{mod}} \cap [\![\mathcal{T}]\!]^{\Delta}_{\mathrm{mod}}$*

The theorem is a direct extension for robust implemenations of Theorem 6 in [1]. We remark that due to the monotonicity of the refinement (Property 1), we can use two different delays Δ_1 and Δ_2, such that $[\![\mathcal{S}]\!]^{\Delta_1}_{\text{mod}} \cap [\![\mathcal{T}]\!]^{\Delta_2}_{\text{mod}} \supseteq [\![\mathcal{S} \wedge \mathcal{T}]\!]^{\max(\Delta_1,\Delta_2)}_{\text{mod}}$. So requirements with different precision can be conjoined, by considering the smaller jitter.

Robustness of the operator in Def. 14 is very fortunate. Thanks to this large parts of implementation of theory of [1] can be reused. We have experimented on small examples asking simple robust consistency questions by applying constructions manually and using non-robust version of ECDAR [15], obtaining promising results.

5 Robust Compatibility, Composition and Quotient

We lift the composition and quotient operators [1] to the robust setting. Composition is used to build systems from smaller units, while quotient is used to synthesize specifications of missing components in a larger design, for example for controller synthesis.

Parallel Composition. Two specifications $\mathcal{S}, \mathcal{T}$ can be composed only iff $Act^{\mathcal{S}}_{\text{o}} \cap Act^{\mathcal{T}}_{\text{o}} = \emptyset$. Parallel composition is obtained by a product, where the inputs of one specification synchronize with the outputs of the other:

Definition 15 (Parallel Composition). *Let $\mathcal{S} = (Loc^S, q_0^S, Clk^S, E^S, Act^S, Inv^S)$ and $\mathcal{T} = (Loc^T, q_0^T, Clk^T, E^T, Act^T, Inv^T)$ be two composable specifications. We define their parallel product, denoted $\mathcal{S} \parallel \mathcal{T}$, as the TIOA $(Loc, q_0, Clk, E, Act, Inv)$ where $Loc = Loc^S \times Loc^T$, $q_0 = (q_0^S, q_0^T)$, $Clk = Clk^S \uplus Clk^T$, $Inv(q_s, q_t) = Inv(q_s) \wedge Inv(q_t)$, and the set of edges is defined by the three following rules:*

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

Robustness distributes over parallel composition in the following fashion:

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

We model incompatibility by introducing a predicate describing undesirable states denoted by the set und. For example, a communication failure in the input-enabled setting can be modeled, by redirecting an input edge to an undesirable location. In general any reachability objective, for example given by a temporal logics 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 [10] 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 16. *A specification $\mathcal{S}$ is* Δ-robust useful *if there exists an environment $\mathcal{E}$ such that no undesirable states are reached in $[\![\lceil \mathcal{E} \rceil^{o}_{\Delta} \parallel \mathcal{S}]\!]_{\text{sem}}$.*

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

Theorem 4. *If there exists a robust winning strategy f in the Δ-robust usefulness game for a specification $\mathcal{S}$, then $\mathcal{S}$ is Δ-robust useful in the environment $\mathcal{E}_f$.*

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

Definition 17. *Two composable specifications $\mathcal{S}$ and $\mathcal{T}$ are* Δ-robust compatible *iff $\mathcal{S} \parallel \mathcal{T}$ is Δ-robust useful.*

It is important that the robust theory does not modify the definition of the operations themself. This means that all the important properties of composition introduced in [1] remain valid. This is illustrated with the independent implementability property in Theorem 5, which follows from Lemma 1 and Thm. 10 in [1].

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

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

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

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

Definition 18 (Quotient). *Let $\mathcal{T} = (Loc^T, q_0^T, Clk^T, E^T, Act^T, Inv^T)$ and $\mathcal{S} = (Loc^S, q_0^S, Clk^S, E^S, Act^S, Inv^S)$ with $Act^S \subseteq Act^T$ and $Act^S_{\text{o}} \subseteq Act^T_{\text{o}}$. Their* quotient*, denoted $\mathcal{T} \setminus\!\setminus \mathcal{S}$, is the TIOA $(Loc, q_0, Clk, E, Act, Inv)$ where $Loc = Loc^T \times Loc^S \cup \{l_{\text{u}}, l_{\emptyset}\}$, $q_0 = (q_0^T, q_0^S)$, $Clk = Clk^T \uplus Clk^S \uplus \{x_{new}\}$, $Act = Act_{\text{i}} \uplus Act_{\text{o}}$ with $Act_{\text{i}} = Act^T_{\text{i}} \cup Act^S_{\text{o}} \cup \{i_{new}\}$ and $Act^T_{\text{o}} \setminus Act^S_{\text{o}}$, $Inv(q_t, q_s) = Inv(l_{\text{u}}) = true$ and $Inv(l_{\emptyset}) = \{x_{new} \leq 0\}$, and the set E of edges is defined by the following rules:*

- $\forall q_t \in \mathit{Loc}^T.\forall q_s \in \mathit{Loc}^S.\forall a \in \mathit{Act}.\exists((q_t,q_s),a,\neg \mathit{Inv}^S(q_s),\{x_{new}\},l_{\mathrm{u}}) \in E$,
- $\forall q_t \in \mathit{Loc}^T.\forall q_s \in \mathit{Loc}^S.\exists((q_t,q_s),i_{new},\neg \mathit{Inv}(q_t) \wedge \mathit{Inv}(q_s),\{x_{new}\},l_{\emptyset}) \in E$,
- *if* $(q_t,a,\varphi_t,c_t,q'_t) \in E^T$ *and* $(q_s,a,\varphi_s,c_s,q'_s) \in E^S$, *then* $\exists((q_t,q_s),a,\varphi^T \wedge \varphi^S, c_t \cup c_s,(q'_t,q'_s)) \in E$,
- $\forall(q_s,a,\varphi_s,c_s,q'_s) \in E^S$ *with* $a \in \mathit{Act}^S_{\mathrm{o}}$, $\exists((q_t,q_s),a,\varphi^S \wedge \neg G^T,\{x_{new}\},l_{\emptyset}) \in E$, *where* $G^T = \bigvee\{\varphi_t \mid (q_t,a,\varphi_t,c_t,q'_t)\}$,
- $\forall(q_t,a,\varphi_t,c_t,q'_t) \in E^T$ *with* $a \notin \mathit{Act}^S$, $\exists((q_t,q_s),a,\varphi^T,c_t,(q'_t,q'_s)) \in E$,
- $\forall(q_t,a,\varphi_t,c_t,q'_t) \in E^T$ *with* $a \in \mathit{Act}^S_{\mathrm{o}}$, $\exists((q_t,q_s),a,\neg G^T,\{\},l_{\mathrm{u}}) \in E$, *where* $G^T = \bigvee\{\varphi_s \mid (q_s,a,\varphi_s,c_s,q'_s)\}$,
- $\forall a \in \mathit{Act}_{\mathrm{i}}.\exists(l_{\emptyset},a,x_{new}=0,\emptyset,l_{\emptyset}) \in E$,
- $\forall a \in \mathit{Act}.\exists(l_{\mathrm{u}},a,true,\emptyset,l_{\mathrm{u}}) \in E$.

As stated in Thm. 12 of [1], the quotient gives a maximal (the weakest) specification for a missing component. This theorem can be generalized to specifications that are locally consistent (see [1]), and used to argue for completeness of the quotient construction in the robust case. It turns out that this very operator is also maximal for the specification of a robust missing component, in the following sense:

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

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

6 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 [1] combined together with a new robust timed game for robust specification theories. We extend the construction of [2] to the setting of specification theories, to solve robust games by reducing them to problems on classical timed games. This construction can easily be implemented in tools such as ECDAR. Our approach 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. Our approach can potentially be applied to lift any game-based timed specification theory to a robust setting.

In the future, we will consider the parametric extension of our theory, that is to synthesize the value of the perturbation for which consistency holds. The emptiness problem that decides if there exists a robust delay has already been studied in the case of 1-player games in different works [6,8,16,17]. A quantitative analysis of this problem has only been studied in [18].

References

1. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: HSCC, ACM (2010) 91–100
2. Chatterjee, K., Henzinger, T.A., Prabhu, V.S.: Timed parity games: Complexity and robustness. In: FORMATS. Volume 5215 of LNCS., Springer (2008) 124–140

3. The COMBEST Consortium: Combest http://www.combest.eu.com.
4. The SPEEDS Consortium: Speeds http://www.speeds.eu.com.
5. Badouel, E., Benveniste, A., Caillaud, B., Henzinger, T., Legay, A., Passerone, R.: Contract theories for embedded systems : A white paper. Research report, IRISA/INRIA Rennes (2009)
6. Puri, A.: Dynamical properties of timed automata. In: Formal Techniques in Real-Time and Fault-Tolerant Systems. Volume 1486 of LNCS. Springer (1998) 210–227
7. Wulf, M.D., Doyen, L., Raskin, J.F.: Almost ASAP semantics: from timed models to timed implementations. Formal Aspects of Computing **17**(3) (2005) 319–341
8. Wulf, M., Doyen, L., Markey, N., Raskin, J.F.: Robust safety of timed automata. Formal Methods in System Design **33** (2008) 45–84
9. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2) (1994) 183–235
10. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC / SIGSOFT FSE. (2001) 109–120
11. de Alfaro, L., Henzinger, T.A.: Interface-based design. In: In Engineering Theories of Software Intensive Systems, Marktoberdorf Summer School. (2004)
12. Bulychev, P., Chatain, T., David, A., Larsen, K.G.: Efficient on-the-fly algorithm for checking alternating timed simulation. In: FORMATS. Volume 5813 of LNCS., Springer (2009) 73–87
13. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: STACS. (1995) 229–242
14. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: CONCUR. Volume 3653 of LNCS., Springer (2005) 66–80
15. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: ECDAR: An environment for compositional design and analysis of real time systems. In: ATVA. Volume 6252 of LNCS., Springer (2010) 365–370
16. Bouyer, P., Markey, N., Reynier, P.A.: Robust model-checking of linear-time properties in timed automata. In: LATIN. Volume 3887 of LNCS., Springer (2006) 238–249
17. Bouyer, P., Markey, N., Reynier, P.A.: Robust analysis of timed automata via channel machines. In: FOSSACS'08. Volume 4962 of LNCS., Springer (2008) 157–171
18. Jaubert, R., Reynier, P.A.: Quantitative robustness analysis of flat timed automata. In: FOSSACS. Volume 6604 of LNCS., Springer (2011) 229–244