A Specification Theory of Real-Time Processes

Chris Chilton[1], Marta Kwiatkowska[1], Faron Moller[2], and Xu Wang[2](✉)

[1] Department of Computer Science, University of Oxford, Oxford, UK
[2] Department of Computer Science, Swansea University, Sketty, UK
xu.wang.comp@gmail.com

AQ1

Abstract. This paper presents an assume-guarantee specification theory (aka interface theory from [11]) for modular synthesis and verification of real-time processes with critical timing constraints. Four operations, i.e. conjunction, disjunction, parallel and quotient, are defined over specifications, drawing inspirations from classic specification theories like refinement calculus [4,19]. We show that a congruence (or pre-congruence) characterised by a trace-based semantics [14] captures exactly the notion of substitutivity (or refinement) between specifications.

AQ2

Dedication: I would like to thank Prof. Bill Roscoe for leading me into the fascinating world of concurrency and nurturing my appreciation for simplicity and elegance in theories of relevance. —— Xu Wang

1 Introduction

Modular synthesis and verification of *quantitative aspects* (e.g. real-time, probability, reward, etc.) of computational and physical processes (e.g. cyber-physical systems) is an important research topic [5].

In this programme of quantitative study, a specification of components consists of a combination of *quantitative assumptions* and *quantitative guarantees.* A refinement relation captures the *substitutability* between quantitative components, adhering to the so-called *contra-variance* principle: refinement implies the relaxation of assumptions as well as the strengthening of guarantees.

As one step of the programme, this paper targets component-based development for real-time systems with critical timing constraints. We propose a complete timed specification theory based on a framework of *minimal extension of timed automata* [1], which is endowed with the operations of *parallel composition* for structural integration, logical *conjunction/disjunction* for viewpoint fusion and independent development, and *quotient* for incremental synthesis. The operations in some sense can be regarded as the concurrent and real-time incarnations of similar operations from refinement calculus [4,16,17,19] (i.e. *sequential composition*, *angelic choice*, *demonic choice*, and *pre-post specification*).

The refinement relation is defined relative to the notion of *incompatibility error* (aka *contract breach* [4]). That is, mismatch of the assumptions and

© Springer International Publishing AG 2017
T. Gibson-Robinson et al. (Eds.): Roscoe Festschrift, LNCS 10160, pp. 1–21, 2017.
DOI: 10.1007/978-3-319-51046-0_2

guarantees between components *composed in parallel* gives rise to errors (aka *abort* [4,19] and denoted $\perp$). Refinement means error-free substitutivity.[1]

Previously, based on this framework, [9] introduced a compositional linear-time specification theory for real-time systems, where the substitutive refinement is the weakest pre-congruence preserving incompatibility errors and characterisable by a finite double trace semantics. A key novelty of [9] lies in the introduction of an explicit *timestop* operation (denoted by $\top$) that halts the progress of the system clock, which, remarkably, corresponds to a timed incarnation of *miracle* or *magic* in refinement calculus[2].

While timestop is appropriate for a restricted class of applications, there are common cases where the operation of stopping the system clock is not meaningful or implementable (aka *infeasible* [19]). Hence, it is desirable to consider systems without explicit or implicit timestops, which we call *realisable systems*.

For realisable systems, components, not substitutively-equivalent according to [9], can become equivalent under realisability due to the environment losing the power to observe the timing difference in error occurrences. Thus, we need a new substitutive equivalence as a coarsening of the congruence in [9].

To best characterise the coarsening, our theory requires a shift of focus to a more game-theoretical treatment, where the coarsening constitutes a reactive synthesis game, called *normalisation*, that collapses erroneous behaviours in a specification. Normalisation is strictly more aggressive than classical timed reactive synthesis [3,7], which enables us to achieve the weakest congruence results.

Furthermore, in a similar vein to timed concurrent games [12,13], where one of the key concerns is the removal of time-blocking strategies by applying blame assignment, the composition of realisable systems (e.g. conjunction or quotient) in our framework generates new unrealisable behaviours, which have to be removed. Rather than employing blame assignment, our framework, reduces the problem to another timed synthesis game that turns out to be precisely the dual game of normalisation called *realisation*, again re-confirming the duality between contract breach and infeasibility of refinement calculus.

Finally our theory presents a trace-semantics characterisation of the refinement and operators, which supports the explicit separation of assumptions and guarantees, and integrates well with automata learning techniques.

Our trace-semantics can be regarded as a timed extension [21] of Dill's trace semantics [14], who first used untimed double-trace semantics for asynchronous circuit verification, i.e. a set of *success traces* and a set of *failure traces*, which, in turn, are inspired by earlier trace theory of asynchronous circuits [15,23] and CSP process algebra [22].

[1] Note that the existence of incompatibility errors does not mean that the composed system is un-usable; an environment can still usefully exploit the system by only utilising the part of the system that is free of the incompatibility errors, as has been well explained in [11].

[2] It were Carroll Morgan and Joseph M. Morris who first added miracle to refinement calculus.

Previously, trace semantics has been the basis of our untimed specification theory [8], which supports all four operators and the weakest congruence preserving substitutivity. We have also connected double-trace semantics with CSP model checking [22] in [26], which can potentially be further extended to connect our timed theory with timed CSP model checking [2].

2 Minimal TA Extension for Timed Specification

Our theory builds on *timed I/O automata* and *timed I/O transition systems.*[3]

2.1 Timed I/O Automata (TIOA)

Clock constraints. Given a set X of real-valued clock variables, a *clock constraint* over X, $cc : CC(X)$, is a boolean combination of atomic constraints of the form $x \bowtie k$ and $x - y \bowtie k$, where $x, y \in X$, $\bowtie \in \{\leq, <, =, >, \geq\}$, and $k \in \mathbb{N}$.

Definition 1. *A* TIOA *is a tuple* $(C, I, O, L, l^0, AT, Inv, coInv)$*, where:*

- $C \subseteq X$ *is a finite set of clock variables (ranged over by* x, y*, etc.)*
- $A = I \uplus O$ *is a finite alphabet (ranged over by* a, b*, etc.) consisting of the input actions* I *and output actions* O
- L *is a finite set of* locations *(ranged over by* l, l', n, n'*, etc.) while* $l^0 \in L$ *is the* initial location
- $AT \subseteq L \times CC(C) \times A \times 2^C \times L$ *is a set of* action transitions
- $Inv : L \to CC(C)$ *and* $coInv : L \to CC(C)$ *assign* invariants *and* co-invariants *to states, each of which is a downward-closed clock constraint.*[4]

In the rest of the paper we use $l \xrightarrow{g,a,rs} l'$ as a shorthand for $(l, g, a, rs, l') \in AT$. $g : CC(C)$ is the enabling guard of the transition, $a \in A$ the action, and rs the subset of clock variables to be reset.

Our TIOAs are an extension of timed automata [1], distinguishing *input from output* and *invariant from co-invariant.* The *semantics of TIOAs* is an extension of timed transition systems (TTSes) called Timed I/O Transition Systems.

2.2 Timed I/O Transition Systems (TIOTSes)

Plain states. A plain state is a pair drawn from $P = L \times \mathbb{R}^C$ (i.e. a location and clock-valuation pair). A *clock valuation* (drawn from $\mathbb{R}^C$) is a map that assigns to each clock variable x in C a real value from $\mathbb{R}^{\geq 0}$.

Definition 2. *A* TIOTS *is a tuple* $\mathcal{P} = \langle I, O, S, s^0, \to\rangle$*.* $S = P \uplus \{\perp, \top\}$ *is a set of states,* $s^0 \in S$ *is the designated initial state, and* $\to \subseteq S \times (I \uplus O \uplus \mathbb{R}^{>0}) \times S$ *is the action- and time-labelled transition relation which is time-additive.*[5]

[3] Our timed framework originally appeared in [9]. However, the version presented here contains important technical extension as well as presentational improvements.

[4] Invariants and guards on output actions are constraints on the *system* (aka guarantees) whereas co-invariants and guards on input actions are constraints on the *environment* (aka assumptions).

[5] $\mathcal{P}$ is *time-additive* providing $p \xrightarrow{d_1+d_2} s'$ iff $p \xrightarrow{d_1} s$ and $s \xrightarrow{d_2} s'$ for some $s \in S$.

Notation. In the rest of the paper we use p, p', p_i to range over P while s, s', s_i range over S. Furthermore we define $tA = I \uplus O \uplus \mathbb{R}^{>0}$, $tI = I \uplus \mathbb{R}^{>0}$, and $tO = O \uplus \mathbb{R}^{>0}$. Symbols like α, β, etc. are used to range over tA.

A *timed trace* (ranged over by tt, tt', tt_i etc.) is a finite mixed sequence of positive real numbers ($\mathbb{R}^{>0}$) and visible actions such that *no two numbers are adjacent to one another.*

For instance, $\langle 0.33, a, 1.41, b, c, 3.1415\rangle$ is a timed trace denoting the observation that action a occurs at 0.33 time units, then another 1.41 time units elapse before the simultaneous occurrence of b and c, which is followed by 3.1415 time units of no event occurrence. The empty trace is denoted by ϵ. An infinite timed trace is an infinite such sequence.

We use $l(tt)$ to indicate the duration of tt, which is obtained as the sum of all the reals in tt, and use $c(tt)$ to count the number of action occurrences along tt. Concatenation of timed traces tt and tt', denoted $tt^\frown tt'$, is obtained by appending tt' onto tt and coalescing adjacent reals (summing them). For instance, $\langle a, 1.41\rangle ^\frown\langle 0.33, b, 3.1415\rangle = \langle a, (1.41 + 0.33), b, 3.1415\rangle = \langle a, 1.74, b, 3.1415\rangle$.

Prefix/extension are defined as usual by concatenation. We write $tt \restriction tA_0$ for the projection of tt onto timed alphabet tA_0, which is defined by removing from tt all actions not inside tA_0 and summing up adjacent reals.

Non-zenoness. For a TIOTS $\mathcal{P}$, we use $p \overset{tt}{\Rightarrow} p'$ to denote a finite execution starting from p that produces trace tt and leads to p'. Similarly, we can define infinite executions which produce infinite traces on $\mathcal{P}$. An infinite execution is *zeno* iff the action count is infinite but the duration is finite.

We say a TIOTS $\mathcal{P}$ is *non-zeno* providing no plain execution is zeno. $\mathcal{P}$ is *strongly non-zeno* iff there exists some $k \in \mathbb{N}$ s.t., for all plain executions $p \overset{tt}{\Rightarrow} p'$, it holds that $l(tt) = 1$ implies $c(tt) \leq k$. Here, we say a finite or infinite execution is a *plain execution* iff the execution only visits plain states.

Assumption on TIOTSs. We only consider non-zeno time-additive TIOTSs in this paper. For technical convenience (e.g. ease of defining time additivity and trace semantics), the definition of TIOTSs requires that $\top$ and $\bot$ are *chaotic states* [22], i.e. a state in which the set of outgoing transitions are all self-loops, one for each $\alpha \in tA$.

The strong non-zenoness is not an assumption of our theory. But with this additional requirement we can show that the synthesis and verification theory in this paper is fully automatable.

2.3 A Game-Based Interpretation

The derivation of TIOTSes from TIOAs is more or less standard, extending the one from TAs to TTSes. Here we just give an intuitive explanation using games. The formal definition can be found in [9].

TIOAs are designed as *mixed assume/guarantee specifications* of timed components. Their semantics is best illustrated by interpreting TIOTSes as timed game graphs. The game has three players: environment, system and coin. The environment controls *input actions* and *delays* while the system controls *output actions* and *delays*. The game has *two game-ending states*: $\top$ (system losing) and $\bot$ (environment losing). States other than $\top$ and $\bot$ are plain states. The coin serves as a tie-breaker for symmetric moves proposed by the other two players.

The environment must respect the constraints on input and delay, i.e. *input guard* and *co-invariant*, which constitutes the assumption half of the specification. *Guard-violating input* and *coinvariant-violating delay* are mapped to $\bot$.

The system must respect the constraints on output and delay, i.e. *output guard* and *invariant*, which constitutes the guarantee half of the specification. *Guard-violating output* and *invariant-violating delay* are mapped to $\top$.

Since delay is controlled by both sides, there exists a contention between invariant and co-invariant violations. If a delay exceeds the *upper bound* of one[6] before exceeding that of the other, the violation of the former will pre-empt the violation of the latter. If a delay *exceeds the upper bounds of both simultaneously*, the invariant violation will be pre-emptive and the delay mapped to $\top$.[7]

On top of game graphs, system and environment, assisted by coin, play a concurrent timed game based on delayed actions:

- A delayed action is either (d, a) or $(\infty, -)$, where $d \in \mathbb{R}^{\geq 0}$ and $a \in I \cup O$.
- Given a current state, each player proposes a delayed action under their control at that state,
  - The delayed action with *strictly smaller delay* will be chosen.
  - If the two delays *tie* (i.e. equal), it will be resolved by *tossing a coin*.
- Fire the chosen delayed action and transit to the destination state.

TIOAs do not have explicit $\top$ and $\bot$. But a $\bot$-location equates to having *true* as invariant and *false* as co-invariant. Dually, we have a $\top$-location. Together, they are reminiscent of abort and magic in their predicate forms [4,19].

2.4 Conventions on Disabled Transitions

In presenting TIOAs and TIOTSes, one often needs to be economical in drawing transitions. So a convention on disabled transitions is required.

1. a disabled input at a plain state is equivalent to an input transition to $\bot$.
2. a disabled output at a plain state is equivalent to an output transition to $\top$.

[6] Note that invariant and co-invariant are downward-closed. Thus, the only way to violate them is to exceed their upper bounds.

[7] One further case missing above is that, for an action transition, there is possibility that its guard is respected but the invariant/co-invariant of its destination (say l) is violated. In such situation, a state (l, t) is treated (1) as $\top$ if t violates the invariant in location l and (2) as $\bot$ if t violates the co-invariant in l while the invariant holds.

Our TIOTSes, on the other hand, disallow *disabled delay transitions*. So the delays enabled at each plain state are *unbounded*, leading to either consistently other plain states or a mixture of plain states with $\top/\bot$ *separated by a finite bound*. The convention induces some semantic-preserving transformations on TIOTSs.

$\top/\bot$ completion. The $\bot$-*completion* of a TIOTS $\mathcal{P}$, denoted $\mathcal{P}^\bot$, adds an a-labelled transition from p to $\bot$ for every $p \in P$ $(= L \times \mathbb{R}^C)$ and $a \in I$ s.t. a is not enabled at p. The $\top$-*completion*, denoted $\mathcal{P}^\top$, adds an a-labelled transition from p to $\top$ for every $p \in P$ and $a \in O$ s.t. a is not enabled at p.

Similarly, we can define $\top/\bot$ completion on TIOAs. We say a TIOA, $\mathcal{P} = (C, I, O, L, n^0, AT, Inv, coInv)$, is $\top$-completed iff, for all $a \in O$ and $l \in L$, we have $\bigvee\{g_k \mid l \xrightarrow{g_k, a, rs_k} l'_k \in AT\} = true$. We say $\mathcal{P}$ is $\bot$-completed iff, for all $a \in I$ and $l \in L$, we have $\bigvee\{g_k \mid l \xrightarrow{g_k, a, rs_k} l'_k \in AT\} = true$.
$\top/\bot$ removal The inverse operations of $\top/\bot$ completion, called $\top/\bot$ *removal*, are also semantic-preserving transformations. For instance, $\top$-removal removes all output transitions from plain states to $\top$ in a TIOTS. We leave it as an exercise for the readers to define $\top/\bot$ removal for TIOAs.

2.5 Liveness and Safety

The constraints in TIOAs can be classified as either *safety* constraints or *liveness* constraints. The former are the guards on transitions while the latter are the invariants/co-invariants on locations.

Example. Fig. 1 depicts a job scheduler together with a printer controller. The invariant at location A of the scheduler forces a *bounded-liveness guarantee* on outputs in that location: as time must be allowed to progress beyond $x = 100$, the *start* action must be fired before x exceeds 100. After *start* being fired, the clock x is reset to 0 and the scheduler waits (possibly indefinitely) for the job to *finish*. If the job finishes, the scheduler expects it to take place at a time point satisfying $5 \le x \le 8$ (i.e. *a safety assumption*).

The controller waits for the job to *start*, after which it will wait exactly 1 time unit before issuing *print* (forced by the invariant $y \le 1$ on state 2 and the guard $y = 1$ on the *print*! transition, acting together as a combined liveness and *safety guarantee*). Then, the controller requires the printer to acknowledge the job as having been *printed* within 10 time units (i.e. co-invariant $y \le 10$ in state 3 acting as *a bounded-liveness assumption*). After receiving it, the controller must indicate to the scheduler, within 5 time units, that the job has *finish*ed.

2.6 Specification Composition: Generic Synchronised Product

This paper introduces a series of four operators for specification composition: $\|$ for parallel composition, $\wedge$ for conjunction, $\vee$ for disjunction and % for quotient.

At the core of these operators is a generic synchronised product $\prod_\otimes$ operation, where $\otimes$ ranges over the set $\{\|, \vee, \wedge, \%\}$. After instantiation, $\prod_\otimes$ produces

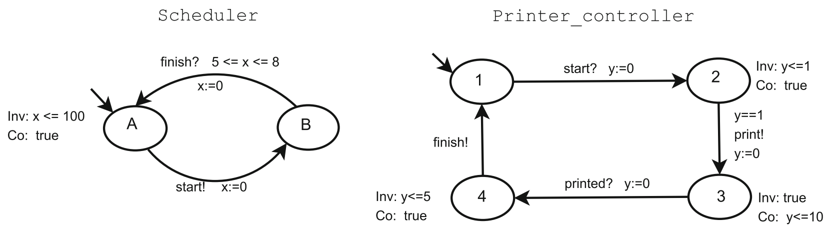

Fig. 1. Job scheduler and printer controller.

four variants ($\prod_{\|}$, $\prod_{\wedge}$, $\prod_{\vee}$ and $\prod_{\%}$), each of which needs further add-on transformations in order to define the four specification composition.

In order to obtain a modular and factored structure, we adopt a two-step approach to defining $\prod_{\otimes}$. In the first step we define, for each $\otimes \in \{\|, \vee, \wedge, \%\}$, a state composition operator $\otimes$ and an alphabet composition operator $\otimes$ (i.e. $\otimes$ is polymorphic). In the second step, we use $\prod_{\otimes}$ to lift the state/alphabet composition to the process composition.

We say (I_0, O_0) and (I_1, O_1) are $\|$-*composable* if $O_0 \cap O_1 = \{\}$, are $\wedge$- *and* $\vee$-*composable* if $(I_0, O_0) = (I_1, O_1)$, and are %-*composable* if (I_0, O_0) *dominate* (I_1, O_1), i.e. $A_1 \subseteq A_0$ and $O_1 \subseteq O_0$. Then, assuming $\otimes$-composability on alphabet pairs, we can define the alphabet composition operations $(I_0, O_0) \otimes (I_1, O_1)$ as follows: $(I_0, O_0) \parallel (I_1, O_1) = ((I_0 \cup I_1) \backslash (O_0 \cup O_1), O_0 \cup O_1)$, $(I_0, O_0) \wedge (I_1, O_1) = (I_0, O_0)$, $(I_0, O_0) \vee (I_1, O_1) = (I_0, O_0)$ and $(I_0, O_0)\%(I_1, O_1) = (I_0 \cup O_1, O_0 \backslash O_1)$.

The definition of $s_0 \otimes s_1$ is supplied in Table 1.[8] Intuitively $\bot$ is equated to an *erroneous specification* while $\top$ is equated to *timestop*, i.e. the operation of stopping the system clock or freezing the global time.

Thus, $\top$ represents the *magic moment* from which the whole system stops running and freezes, eliminating, once and for all, all subsequent possibility of reaching the erroneous state. This gives rise to a *refinement ordering* over states, whereby $\top$ refines plain states, which in turn refine $\bot$.

Timestop can explain the behaviour of $\top$ in parallel composition: the equation $\bot \parallel \top = \top$ holds because time stops exactly at the moment of reaching the erroneous state, so the resulting state is a timestop, rather than $\bot$.

It is also easy to see that *state conjunction* ($\wedge$) and *disjunction* ($\vee$) operations in Table 1 follow the intuition of the join and meet operations.

The *state quotient* (%) operation is harder to explain. But some intuition can be recovered from the *derivation* of % based on $\|$ and $\neg$, i.e. $s_0\%s_1 = (s_0^{\neg} \parallel s_1)^{\neg}$, where *state mirror* ($\neg$) behaves like negation (c.f. Table 1).

State-to-process lifting. Given two $\top/\bot$ completed TIOTS, $\mathcal{P}_i = \langle I_i, O_i, S_i, s_i^0, \rightarrow_i \rangle$ for $i \in \{0, 1\}$, s.t. $S_0 \cap S_1 = \{\bot, \top\}$ and (I_0, O_0) and (I_1, O_1) are $\otimes$-composable, $\mathcal{P}_0 \prod_{\otimes} \mathcal{P}_1$ gives rise to a new $\top/\bot$ completed TIOTS $\mathcal{P} = \langle I, O, S, s^0, \rightarrow \rangle$ s.t. $(I, O) = (I_0, O_0) \otimes (I_1, O_1)$, $S = (P_0 \times P_1) \uplus P_0 \uplus P_1 \uplus \{\top, \bot\}$,

[8] For $i \in \{0, 1\}$ and $p_i = (l_i, t_i)$, $p_0 \times p_1 = ((l_0, l_1), t_0 \uplus t_1)$ ($t0$ and t_1 are clock-disjoint).

Table 1. State composition operators.

$\parallel$	$\top$	p_0	$\bot$
$\top$	$\top$	$\top$	$\top$
p_1	$\top$	$p_0{\times}p_1$	$\bot$
$\bot$	$\top$	$\bot$	$\bot$

$\wedge$	$\top$	p_0	$\bot$
$\top$	$\top$	$\top$	$\top$
p_1	$\top$	$p_0{\times}p_1$	p_1
$\bot$	$\top$	p_0	$\bot$

$\vee$	$\top$	p_0	$\bot$
$\top$	$\top$	p_0	$\bot$
p_1	p_1	$p_0{\times}p_1$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\%$	$\top$	p_0	$\bot$
$\top$	$\bot$	$\bot$	$\bot$
p_1	$\top$	$p_0{\times}p_1$	$\bot$
$\bot$	$\top$	$\top$	$\bot$

$\neg$	
$\top$	$\bot$
p	p
$\bot$	$\top$

$s^0 = s_0^0 \otimes s_1^0$ and $\rightarrow$ is the smallest relation containing $\rightarrow_0 \cup \rightarrow_1$,[9] and satisfying the rules:

$$\frac{p_0 \xrightarrow{\alpha}_0 s_0' \quad p_1 \xrightarrow{\alpha}_1 s_1'}{p_0 \otimes p_1 \xrightarrow{\alpha} s_0' \otimes s_1'} \qquad \frac{p_0 \xrightarrow{a}_0 s_0' \quad a \notin A_1}{p_0 \otimes p_1 \xrightarrow{a} s_0' \otimes p_1} \qquad \frac{p_1 \xrightarrow{a}_1 s_1' \quad a \notin A_0}{p_0 \otimes p_1 \xrightarrow{a} p_0 \otimes s_1'}$$

Remark. Note the subtlety in the transition rules of $\mathcal{P}_0 \prod_\wedge \mathcal{P}_1$ and $\mathcal{P}_0 \prod_\vee \mathcal{P}_1$. If we have $p_0 \xrightarrow{\alpha} p_0'$ in $\mathcal{P}_0$ and $p_1 \xrightarrow{\alpha} \top$ in $\mathcal{P}_1$, then we have $p_0 \times p_1 \xrightarrow{\alpha} p_0'$ in $\mathcal{P}_0 \prod_\wedge \mathcal{P}_1$. That is, process $\mathcal{P}_1$ is discarded after the transition and the rest of the execution is the solo run of $\mathcal{P}_0$.[10]

We can also lift the state mirror operator $\neg$ to process level by defining the *pre-mirror* operator $\neg_0$; $\mathcal{P}^{\neg_0}$ interchanges $I_\mathcal{P}$ and $O_\mathcal{P}$ as well as $\top$ and $\bot$ in $\mathcal{P}$.

The definition of parallel synchronised product can be lifted to TIOAs. Given two $\otimes$-composable $\top/\bot$-completed TIOAs with disjoint clocks ($C_0 \cap C_1 = \{\}$), $\mathcal{P}_i = (C_i, I_i, O_i, L_i, n_i^0, AT_i, Inv_i, coInv_i)$ for $i \in \{0,1\}$, their synchronised product gives rise to another TIOA $\mathcal{P} = \mathcal{P}_0 \prod_\otimes \mathcal{P}_1$:

- $C = C_0 \cup C_1$, $(I, O) = (I_0, O_0) \otimes (I_1, O_1)$, $L = L_0 \times L_1$ and $n^0 = n_0^0 \times n_1^0$;
- AT is the least relation that contains AT_0, AT_1 and $\{l_0 \times l_1 \xrightarrow{g_0 \wedge g_1, a, rs_0 \cup rs_1} n_0' \times n_1' \mid l_0 \xrightarrow{g_0, a, rs_0} n_0' \in AT_0 \wedge l_1 \xrightarrow{g_1, a, rs_1} n_1' \in AT_1\}$
  $\cup \{l_0 \times l_1 \xrightarrow{g_0, a, rs_0} n_0' \times l_1 \mid l_0 \xrightarrow{g_0, a, rs_0} n_0' \in AT_0, a \in (A_0 \backslash A_1)\}$
  $\cup \{l_0 \times l_1 \xrightarrow{g_1, a, rs_1} l_0 \times n_1' \mid l_1 \xrightarrow{g_1, a, rs_1} n_1' \in AT_1, a \in (A_1 \backslash A_0)\}\}$;
- and $(Inv(l_0 \times l_1), coInv(l_0 \times l_1)) = (Inv_0(l_0), coInv_0(l_0)) \otimes (Inv_1(l_1), coInv_1(l_1))$.

We define the invariant/co-invariant composition operation $\otimes$ as follows[11]:

- $(Inv_0, coInv_0) \parallel (Inv_1, coInv_1) = (Inv_0 \wedge Inv_1, coInv_0 \wedge coInv_1)$
- $(Inv_0, coInv_0) \wedge (Inv_1, coInv_1) = (Inv_0 \wedge Inv_1, coInv_0 \vee coInv_1)$
- $(Inv_0, coInv_0) \vee (Inv_1, coInv_1) = (Inv_0 \vee Inv_1, coInv_0 \wedge coInv_1)$
- $(Inv_0, coInv_0) \% (Inv_1, coInv_1) = (Inv_0 \wedge coInv_1, coInv_0 \wedge Inv_1)$

The pre-mirror ($\mathcal{P}^{\neg_0}$) of a TIOA $\mathcal{P}$ interchanges $I_\mathcal{P}$ and $O_\mathcal{P}$ as well as the invariant and co-invariant for each location of $\mathcal{P}$.

[9] Containment of $\rightarrow_0 \cup \rightarrow_1$ is not required for parallel composition, but is necessary for conjunction and disjunction.

[10] The technique was inspired by a discussion with Roscoe on angelic choice in CSP.

[11] Note that the above definition exploits the fact that the addition or removal of *false*-guarded transitions to AT will not change the semantics of the automata.

3 Parallel Composition, Refinement and Determinisation

We define the parallel composition of specifications as $\mathcal{P}_0 \parallel \mathcal{P}_1 = \mathcal{P}_0^\perp \prod_{||} \mathcal{P}_1^\perp$, since $\prod_{||}$ can be extended without modification to work on $\perp$-complete TIOTSs.

Informally, we say one specification is a refinement of another if the former can replace the latter in all closed contexts. A *closed context* of a specification $\mathcal{P}$ is another specification $\mathcal{Q}$ s.t. (1) $\mathcal{P}$ and $\mathcal{Q}$ are $||$-composable and (2) $I_\mathcal{P} \subseteq O_\mathcal{Q} \wedge I_\mathcal{Q} \subseteq O_\mathcal{P}$.

Definition 3 (Substitutive Refinement [9]). *Let $\mathcal{P}_{imp}$ and $\mathcal{P}_{spec}$ be TIOTSs with identical alphabets. We say $\mathcal{P}_{spec} \sqsubseteq \mathcal{P}_{imp}$ iff for all closed contexts $\mathcal{Q}$, $\mathcal{P}_{spec} \parallel \mathcal{Q}$ is $\perp$-free implies $\mathcal{P}_{imp} \parallel \mathcal{Q}$ is $\perp$-free. We say $\mathcal{P}_{spec} \simeq \mathcal{P}_{imp}$ (i.e.* substitutively equivalent*) iff $\mathcal{P}_{imp} \sqsubseteq \mathcal{P}_{spec}$ and $\mathcal{P}_{spec} \sqsubseteq \mathcal{P}_{imp}$.*

A first observation of the refinement definition is that each specification has a deterministic counterpart to which it is substitutively equivalent. The counterpart can be constructed by a modified determinisation procedure.

Determinism. A TIOTS is *deterministic* iff there is no ambiguous transition, i.e. $s \xrightarrow{\alpha} s' \wedge s \xrightarrow{\alpha} s''$ implies $s' = s''$. A TIOA is *deterministic* iff, for each $l \in L$ and $a \in A$, l has a pair of distinct a-transitions $l \xrightarrow{g_1, a, rs_1} l_1$ and $l \xrightarrow{g_2, a, rs_2} l_2$ implies g_1 and g_2 are disjoint.

We define the *determinisation* $\mathcal{P}^D$ of $\mathcal{P}$ as a modified subset construction procedure on $\mathcal{P}^\perp$: given a subset S_0 of states reachable by a given trace, we only keep those which are minimal w.r.t. the state refinement ordering.[12]

Proposition 1 ([9]). *Any TIOTS $\mathcal{P}$ is substitutively equivalent to the deterministic TIOTS $\mathcal{P}^D$.*

From a game theoretical perspective, our modified determinisation procedure converts an imperfect-information game into a perfect-information game.

On the level of TIOAs, strongly non-zeno TAs are known to be determinisable with a symbolic procedure [6], based on which we can implement our procedure (say $DET(\mathcal{P})$) to determinise TIOA $\mathcal{P}$.

In the sequel we focus on deterministic TIOA/TIOTS, i.e. *interfaces*.

4 A Story of Two Games

Our realisability theory will build on a pair of two-player games dual to each other: *normalisation* and *realisation*. They are derivatives of the three-player game in Sect. 2. In all three games, the system tries to steer the game play clear of $\top$ while the environment tries to steer clear of $\perp$.

We give the technical definition of the games in this section, deferring the provision of intuition and *their uses in specification theories* to the next section.

[12] The modified determinisation procedure first appeared in the Definition 4.2 of [26], which is for the untimed case.

4.1 Timed Strategies

An interface $\mathcal{P}$, being a game graph, encodes a set of strategies for each of the three players. We give a formal definition of (timed) strategies below:

– A *system strategy* $\mathcal{G}_s$ is a deterministic tree TIOTS[13] s.t. each plain state p in $\mathcal{G}_s$ is ready to accept all possible inputs by the environment (i.e. a is enabled for all $a \in I$), but allows a single move by the system.
  The system move (denoted $mv(p)$) can be a delayed output (d, a) for some $a \in O$ and $d \in \mathbb{R}^{\geq 0}$ or an infinite delay $(\infty, -)$.[14]
  Dually, we can define *environment strategies* (e.g. $\mathcal{G}_e$). A system strategy is a $\bot$-complete TIOTS while an environment strategy is $\top$-complete.
– Given TIOTSs $\mathcal{P}$ and $\mathcal{P}'$ with *identical alphabets* (i.e. $O = O'$ and $I = I'$), we say $\mathcal{P}$ is a *partial unfolding* [25] of $\mathcal{P}'$ if there exists a function $f : S_{\mathcal{P}} \rightarrow S_{\mathcal{P}'}$ such that (1) f maps $\top$ to $\top$, $\bot$ to $\bot$ and plain states to plain states, and (2) $f(s^0_{\mathcal{P}}) = s^0_{\mathcal{P}'}$ and $p \xrightarrow{\alpha}_{\mathcal{P}} s \Rightarrow f(p) \xrightarrow{\alpha}_{\mathcal{P}'} f(s)$.
– We say a TIOTS $\mathcal{P}$ *contains* a strategy $\mathcal{G}$, denoted $\mathcal{G} \in \mathcal{P}$, if $\mathcal{G}$ is a partial unfolding of $(\mathcal{P}^{\bot})^{\top}$. We say there is a strategy $\mathcal{G}$ *at state* p in $\mathcal{P}$, if $\mathcal{G} \in \mathcal{P}(p)$, where $\mathcal{P}(p)$ is the TIOTS $\mathcal{P}$ re-initialised to state p.

The coin is treated as a special player. A strategy of the coin is a function h from tA^* to $\{0, 1\}$. We denote the set of all possible coin strategies as H.

Strategy composition. A composition of a set of three strategies, denoted $\mathcal{G}_s \times_h \mathcal{G}_e$, will produce, according to the timed concurrent game rules defined in Sect. 2, a simple path which is a partial unfolding of both $\mathcal{G}_s$ and $\mathcal{G}_e$. The simple path can be either *finite and ending in* $\top/\bot$ or *infinite*.

4.2 Two Games

Normalisation game. In the normalisation game, the system forms a coalition with the coin to play against the environment and seek $\bot$-reachability.

Given an interface $\mathcal{P}$, we say a system strategy $\mathcal{G}$ at p and a coin strategy $h \in H$ is *winning* at p iff $\mathcal{L} \times_h \mathcal{G}$ ends in $\bot$ for all possible environment strategies $\mathcal{L}$ at p. Then we say a plain state p in $\mathcal{P}$ is $\bot$-*winning* iff the system and the coin have a winning strategy at p.

[13] We say an acyclic TIOTS is a *tree* if (1) there does not exist a pair of transitions in the form of $p \xrightarrow{a} p''$ and $p' \xrightarrow{d} p''$, (2) $p \xrightarrow{a} p'' \wedge p' \xrightarrow{b} p''$ implies $p = p'$ and $a = b$ and (3) $p \xrightarrow{d} p'' \wedge p' \xrightarrow{d} p''$ implies $p = p'$.

[14] For the former, $\mathcal{G}_s$ generates exactly a time interval $(0, d]$ of delays from p, after which $\mathcal{G}_s$ arrives at another plain state with a enabled. For the latter, an infinite time interval $(0, \infty)$ of delays are enabled at p. The delays either all lead to plain states or $(0, \infty)$ can be further partitioned into two intervals s.t. the delays in the first interval lead to plain states while those of the second lead to $\top$ or $\bot$.

Conversely, we say an environment strategy $\mathcal{L}$ at p is a *normalising strategy*[15] at p iff $\mathcal{L}$ from p can steer the game play clear of $\bot$, i.e. for all coin strategies $h \in H$ and system strategies $\mathcal{G}$ at p, $\mathcal{L} \times_h \mathcal{G}$ produces either a finite play ending in $\top$ or an infinite play.

Interestingly, an environment strategy is normalising iff it is *normalisable*, i.e. it is free of $\bot$. Thus, a state is a $\bot$-winning state iff it contains no normalisable (or normalising) environment strategy.

Synthesis of game winning states is a central problem of the game-theoretical research. To synthesise $\bot$-winning states in interfaces, we focus on the two *representative subclasses* of $\bot$-winning states: auto-$\bot$ and semi-$\bot$ states.

Auto-$\bot$ and semi-$\bot$. Given a $\top/\bot$ complete interface, we say a plain state p is an *auto-$\bot$ state* iff $p \xrightarrow{a} \bot$ for some $a \in O$. We say a plain state p is a *semi-$\bot$ state* iff (1) all input transitions in p or any of its time-passing successors lead to $\bot$, and (2) there exists $d \in \mathbb{R}^{>0}$ s.t. $p \xrightarrow{d} \bot$. For a general interface $\mathcal{P}$, we say p is an auto-$\bot$ (or semi-$\bot$) state in $\mathcal{P}$ iff it is an auto-$\bot$ (or semi-$\bot$) state in $(\mathcal{P}^\top)^\bot$.

For auto-$\bot$ and semi-$\bot$ states, system (and coin) has a one-step winning strategy to reach $\bot$, which are a delay move and an output move resp. The absence of semi-$\bot$/auto-$\bot$ states characterises the absence of $\bot$-winning states.

Lemma 1. *An interface is free of $\bot$-winning states iff it is free of semi-$\bot$ and auto-$\bot$ states.*

Hence we can *find and remove* all $\bot$-winning states in an interface by finding and removing all auto-$\bot$ and semi-$\bot$ states in it.

Normalisation. The normalisation of an interface $\mathcal{P}$, denoted $\mathcal{P}^N$, is obtained by collapsing all $\bot$-winning states in $\mathcal{P}$ to $\bot$, which can be implemented by a $\bot$-*backpropagation* procedure that repeatedly collapses semi-$\bot$ and auto-$\bot$ states in $\mathcal{P}$ to $\bot$, until semi-$\bot$ and auto-$\bot$ freedom is obtained. Normalisation returns a *normalised interface*, which is either the $\bot$-*TIOTS* (i.e. a degenerated TIOTS with $\bot$ as the initial state) or a TIOTS free of $\bot$-winning states.

On deterministic TIOAs, we can implement $\bot$-backpropagation procedures by fixpoint calculation via constraint backpropagation (based on weakest precondition calculation), denoted as $BP(\mathcal{P}, \bot)$.

Realisation Game. In the realisation game, the environment forms a coalition with the coin to play against the system and seek $\top$-reachability. By duality we obtain the definition of $\top$-*winning*, auto-$\top$ and semi-$\top$ states.[16]

[15] We choose not to call it a winning strategy as it serves additional purpose for our paper.

[16] Given a $\top/\bot$ complete interface, we say a plain state p is an *auto-$\top$* iff $p \xrightarrow{a} \top$ for some $a \in I$; a plain state p is a *semi-$\top$* iff (1) all output transitions in p or any of its time-passing successors lead to the $\top$ state, and (2) there exists $d \in \mathbb{R}^{>0}$ s.t. $p \xrightarrow{d} \top$.

We say a system strategy is *realising* iff it can steer the realisation game play clear of $\top$, which is equivalent to being *realisable*, i.e. free of $\top$. Obviously a state is a $\top$-winning state iff it contains no realisable or realising system strategy.

Lemma 2. *An interface is free of* $\top$*-winning states iff it is free of semi-*$\top$ *and auto-*$\top$ *states.*

Similarly we can *find and remove* all $\top$-winning states in an interface by a *realisation* operation.

Realisation. The realisation of an interface $\mathcal{P}$, denoted $\mathcal{P}^R$, is obtained by collapsing all $\top$-winning states in $\mathcal{P}$ to $\top$ (implementable by a dual $\top$-*backpropagation* procedure on TIOTSes or a constraint-backpropagation procedure $BP(\mathcal{P}, \top)$). Realisation returns a *realised interface*, which is either the $\top$-*TIOTS* (i.e. with $\top$ as the initial state) or a TIOTS free of $\top$-winning states.

Interference between the two games. Note that a state in an interface can be simultaneously $\bot$- and $\top$-winning (e.g. simultaneously auto-$\top$ and auto-$\bot$). The anomaly arises due to the coin being shared by both coalitions.

Since coin can only be on one side at a time, this implies that the two games must be played *one-at-a-time* rather than simultaneously.

Hence, in our realisability theory it is meaningless to have states that are both $\bot$- and $\top$-winning. In the sequel we will apply realisation and normalisation operations *alternatingly* to ensure all generated interfaces are *well-formed*, i.e. having no state simultaneously $\bot$- and $\top$-winning.

We say a state is a *neutral states* iff it is neither $\top$-winning nor $\bot$-winning. An interface free of $\top$-winning and $\bot$-winning states is called a *neutral interface.*

The *fundamental principle of interfaces* is to *ensure that all interactions between the system and environment stay in neutral states.*

5 Realisable Specification Theory

When a component, specified by an interface $\mathcal{P}$, interacts with an environment, it plays a game with the environment. This game on a closer look, however, is not identical to the game defined (on the game graph $\mathcal{P}$) in Sect. 4. The component strategies in the new game is still constrained by $\mathcal{P}$ (i.e. as it is for the system strategies contained by $\mathcal{P}$ in the old game). But the environment is entirely un-constrained, which may choose from all strategies definable by its alphabet. Thus, the environment can be extremely powerful in such game interactions, especially when it is further equipped with the timestop operation.

Previously [9], we have developed a specification theory for such systems, where $\simeq$ gives rise to a weakest congruence w.r.t. $\prod_{\|}$, $\prod_{\wedge}$, $\prod_{\vee}$ and $\prod_{\%}$ operations of this paper. It results in a greatly simplified theory without the need for timed game synthesis.

In this section we are going to remove the timestop and its related time-blocking behaviours from both components and environments, and develop a new specification theory for realisable components.

For a proper treatment of un-constrained strategies, we need first to define a notion of *aggressiveness.*

Comparing strategies. Different strategies vary in their effectiveness to steer the interaction clear of $\top$ or $\bot$. Such effectiveness can be compared if two strategies closely resemble each other: we say $\mathcal{G}$ and $\mathcal{G}'$ of the same player are *affine* if $s^0_{\mathcal{G}} \overset{tt}{\Rightarrow} p$ and $s^0_{\mathcal{G}'} \overset{tt}{\Rightarrow} p'$ implies $mv_{\mathcal{G}}(p) = mv_{\mathcal{G}'}(p')$. Intuitively, this means $\mathcal{G}$ and $\mathcal{G}'$ propose the same move at the 'same' states.

Given two affine strategies $\mathcal{G}$ and $\mathcal{G}'$, we say $\mathcal{G}$ is *more $\bot$-aggressive* than $\mathcal{G}'$, denoted $\mathcal{G} \preceq \mathcal{G}'$, if (1) $s^0_{\mathcal{G}'} \overset{tt}{\Rightarrow} \bot$ implies there is a prefix tt_0 of tt s.t. $s^0_{\mathcal{G}} \overset{tt_0}{\Longrightarrow} \bot$ and (2) $s^0_{\mathcal{G}} \overset{tt}{\Rightarrow} \top$ implies there is a prefix tt_0 of tt s.t. $s^0_{\mathcal{G}'} \overset{tt_0}{\Longrightarrow} \top$. Intuitively, it means $\mathcal{G}$ can reach $\bot$ faster but $\top$ slower than $\mathcal{G}'$. $\preceq$ forms a partial order over the set of strategies possessed by a player. Dually, we can define $\mathcal{G}$ being *more $\top$-aggressive* than $\mathcal{G}'$ as $\mathcal{G}' \preceq \mathcal{G}$.

'Representative' winning strategies. We say an environment strategy $\mathcal{G}_e$ is a *winning strategy* in the interaction with component $\mathcal{P}$ iff $\mathcal{G}_s \times_h \mathcal{G}_e$ does not end in $\bot$ for all coin strategies h and all system strategies $\mathcal{G}_s \in \mathcal{P}$.

Of all environment winning strategies against component $\mathcal{P}$, the subset of minimally $\top$-aggressive ones can *fully represent* the whole set (by an upward-closure operation on $\preceq$), since the capability of a less aggressive strategy in steering clear of $\bot$ implies the same capability for more aggressive ones.

Thus, our theory can focus mainly on 'representative' environment winning strategies, which, by the magic of mirror, have already been encoded in $\mathcal{P}$.

Lemma 3. *$\mathcal{G}_e$ is a minimally $\top$-aggressive environment winning strategy in the game with component $\mathcal{P}$ iff $\mathcal{P}^{\neg_0}$ (i.e. pre-mirror of $\mathcal{P}$) contains $\mathcal{G}_e$.*

In another word, an interface $\mathcal{P}$ encodes both a set of component strategies (say SG) and a 'representative' set of environment winning strategies (EG), which are resp. the component guarantees and environment assumptions of the interface.

5.1 Unrealisability

The timestop operation $\top$ freezes the global time by halting the progress of the system clock. In general, such capability is too powerful to be realistic. Thus, a (component or environment) strategy containing timestop is *unrealisable*, and a state possessing no realisable component strategy is an *unrealisable state.*

According to Sect. 4, unrealisable states are exactly $\top$-winning states. Realisation operation is equivalent to removing all unrealisable system strategies from an interface.

Lemma 4. *Given an interface $\mathcal{P}$, the set of realisable component strategies of $\mathcal{P}$ is exactly the set of component strategies of $\mathcal{P}^R$.*

5.2 Incompatibility

Given two interfaces $\mathcal{P}$ and $\mathcal{Q}$ with *complementary* alphabets (i.e. I and O interchanged), their parallel composition calculates the intersection of $SG_{\mathcal{P}}$ and $EG_{\mathcal{Q}}$ as well as that of $SG_{\mathcal{Q}}$ and $EG_{\mathcal{P}}$. That is, the guarantees provided by one interface will be matched against the assumptions required by the other.

For a general component $\mathcal{P}$, both $SG_{\mathcal{P}}$ and $EG_{\mathcal{P}}$ may contain unrealisable strategies. For a realisable component $\mathcal{P}$, only $EG_{\mathcal{P}}$ may contain unrealisable strategies. In a specification theory, environments are also components. If all components are realisable, the unrealisable part of $EG_{\mathcal{P}}$ becomes irrelevant. For instance, if $EG_{\mathcal{P}}$ consists of only unrealisable strategies, it is equivalent to being empty.

In the process of fulfilling assumptions with guarantees, if there is a match (i.e. non-empty realisable intersection), assumptions will be absorbed by guarantees and disappear since $\mathcal{P} \parallel \mathcal{Q}$ forms a closed system. Otherwise (i.e. empty realisable intersection), it gives rise to the so-called *incompatible states*, i.e. states in which all 'representative' environment winning strategy are unrealisable.

A state p in $\mathcal{P}$ is incompatible implies p in $\mathcal{P}^{\neg_0}$ is unrealisable, which in turn implies (by duality) p in $\mathcal{P}$ is *un-normalisable*, i.e. a state containing no normalisable environment strategy. According to Sect. 4, un-normalisable states are exactly $\perp$-winning states.

In assume-guarantee specification theories, auto-$\perp$ and semi-$\perp$, as members of incompatible states, are endowed with specialised interpretations, capturing resp. safety mismatch errors (aka *exception*) and liveness mismatch errors (aka *time-out*).

Exception. The arrival of an input at a location and time of a component when it is not expected (i.e. the input is disabled at the location and time) triggers an exception in the parallel composition. Exception is captured by auto-$\perp$ states.

Figure 2 shows the parallel composition of the job scheduler with the printer controller. In the transition from $B4$ to $A1$, the guard combines the effects of the constraints on the clocks x and y. As *finish* is an output of the controller, it can be fired at a time when the scheduler is not expecting it, meaning that an exception is raised due to safety errors. This is indicated by the transition to $\perp$ when the guard constraint $5 \leq x \leq 8$ is not satisfied.

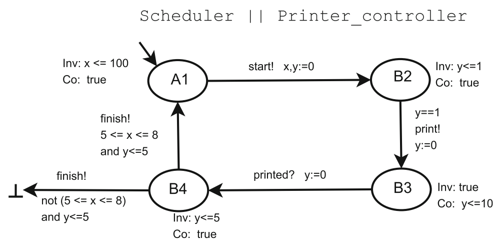

Fig. 2. Parallel composition of the job scheduler and printer controller.

Timeout. The non-arrival of an expected input at a location of a component before the expiration of its co-invariant triggers a *bounded-liveness error* (aka timeout) in the parallel composition.

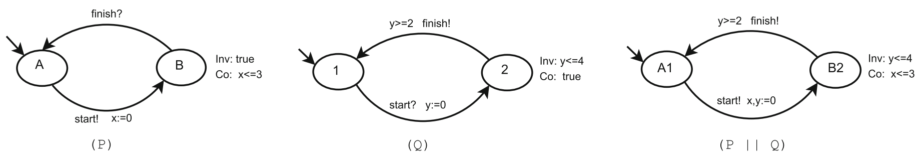

Fig. 3. Bounded liveness error.

Figure 3 shows an example for bounded-liveness errors. In the closed system $\mathcal{P} \parallel \mathcal{Q}$, at location $B2$ the system is free to choose either output *finish* after $y \geq 2$ or delay until $x > 3$. If it chooses the latter, $\mathcal{P}$ component will time out in location B and the system will enter $\bot$. Note that the timeout here is due to the fact that the urgency requirement at location 2 of $\mathcal{Q}$ (i.e. $y <= 4$) is weaker than the timeout bound set at location B of $\mathcal{P}$ (i.e. $x <= 3$). (If it is otherwise, the invariant at $B2$ will preempt the co-invariant at $B2$ and eliminate the possibility of timeout.)

5.3 Realisable Specification and Coarsened Refinement

Now let us start to move back to specifications by defining realisable specifications, which will give us the advantage of the closure under hiding and renaming operations.[17]

We first notice that the definition of auto-$\top$ and semi-$\top$ can be extended to specifications. Then we say a specification is *realisable* iff it is free of both auto-$\top$ and semi-$\top$. Due to the preservation of auto-$\top$ and semi-$\top$ freedom by determinisation, we have:

Lemma 5. *Given a realisable specification $\mathcal{P}$, $\mathcal{P}^D$ is a realisable interface.*

Recall that $\mathcal{P}$ and $\mathcal{P}^D$ are substitutively equivalent according to the finest $\simeq$, in which the timestop operations greatly increase the distinguishing power of the processes, enabling it to tell two interfaces apart by examining the *timing difference* in error occurrences as well as the *existence* of such occurrences.[18]

After the removal of timestop and restricting to realisable specifications, however, the substitutive equivalence is coarsened to be $\simeq_r$.

[17] We omit the two operators in this paper due to space limitation.

[18] That is, they can distinguish the $\bot$ state from the $\bot$-winning states by stopping time immediately.

Realisable refinement. Let $\mathcal{P}$ and $\mathcal{Q}$ be realisable specifications with identical alphabets. $\mathcal{P}$ *realisably refines* $\mathcal{Q}$ (i.e. $\mathcal{Q} \sqsubseteq_r \mathcal{P}$), iff, for all realisable specification $\mathcal{R}$ that is a closed context of $\mathcal{P}$, $\mathcal{Q} \parallel \mathcal{R}$ is $\perp$-free implies $\mathcal{P} \parallel \mathcal{R}$ is $\perp$-free. We say $\mathcal{Q} \simeq_r \mathcal{P}$ (*realisably equivalent*), iff $\mathcal{P} \sqsubseteq_r \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq_r \mathcal{P}$.

It is obvious that $\simeq_r$ is the weakest equivalence preserving incompatible states (over realisable specifications). In the sequel we show that $\simeq_r$ is a congruence w.r.t. the parallel $\parallel$, conjunction $\wedge$, disjunction $\vee$ and quotient % operators.

Note that, even though the sequel focuses on realisable specifications which are closed under all four operations, we still need unrealisable specifications as a detour to simplify operator definitions like quotient and conjunction, since realisable specifications are not closed under $\prod_\wedge$ and $\prod_\%$.

Lemma 6. *Given a realisable specification $\mathcal{P}$, $\mathcal{P} \simeq_r \mathcal{P}^D \simeq_r (\mathcal{P}^D)^N$.*

6 Conjunction, Disjunction and Quotient

In this section we will present the operational definition of conjunction, disjunction and quotient operators[19], building on top of the generic synchronised product operator in Sect. 2.

Desiderata of the operators. Let us first describe the desired effects these operators aim to achieve before the formal development.

Over the set of realisable specifications, e.g. $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{L}$, and with respect to the substitutive refinement $\preceq_r$: (1) $\mathcal{P} \vee \mathcal{Q}$ gives rise to the *strongest* realisable specification that are *weaker than both* $\mathcal{P}$ *and* $\mathcal{Q}$; (2) $\mathcal{P} \wedge \mathcal{Q}$ gives rise to the *weakest* realisable specification that are *stronger than both* $\mathcal{P}$ *and* $\mathcal{Q}$; and (3) $\mathcal{P}\%\mathcal{Q}$ gives rise to the *weakest* realisable specification $\mathcal{L}$ s.t. $\mathcal{L} \parallel \mathcal{Q}$ *is stronger than* $\mathcal{P}$.

Thus, conjunction and disjunction calculate the meet and join w.r.t. $\preceq_r$, whilst quotient synthesises a controller to interact with the specification and steer its execution away from incompatible states.

Operational definitions. The definition of $\prod_\wedge$ can be extended without modification to work on $\perp$-complete TIOTSs.[20] The definitions of $\prod_\vee$ and $\prod_\%$ do not extend to $\perp$-complete TIOTSs.

We define $\mathcal{P} \vee \mathcal{Q} = \mathcal{P}^\top \prod_\vee \mathcal{Q}^\top$ and $\mathcal{P}^\neg = ((\mathcal{P}^D)^N)^{\neg_0}$. We define the other two operators by a three-step recipe: $(((\mathcal{P}^D)^N)^\top \prod_\otimes ((\mathcal{Q}^D)^N)^\top)^R$. We start with normalisation, go on with applying the $\prod_\otimes$ operators (after $\top$-completion), and finish with realisation. It is easy to verify that realisable specifications are closed under all three operators.

We can verify that $\mathcal{P}_0\%\mathcal{P}_1 \simeq_r (\mathcal{P}_0^\neg \parallel \mathcal{P}_1)^\neg$. This is a lifting, from the state level to the process level, of a corresponding equation in Sect. 2.

[19] It is easy to verify that realisable specifications are closed under $\parallel$ defined in Sect. 3 since $\parallel$ preserves auto-$\top$ and semi-$\top$ freedom.

[20] With the extension, blocked synchronisation, i.e. an action being enabled on one process but not so on the other, becomes possible.

7 Declarative Theory of Contracts

We now present a timed-trace semantics to all the operators defined in this paper. For this purpose we adopt the *contract* framework promoted in [5,20][21], which has the advantage of explicitly separating *assumptions* from *guarantees.*

Given a specification $\mathcal{P} = \langle I, O, S, s^0, \rightarrow \rangle$, three sets of traces can be extracted from $((\mathcal{P}^{\perp})^{\top})^{D}$:

- TP is a set of timed traces leading to plain states
- TF is a set of timed traces leading to the erroneous state $\perp$
- TM is a set of timed traces leading to the timestop state $\top$.

TF and TM are extension-closed due to the chaotic nature of $\top$ and $\perp$, while TP is prefix-closed. Since $TF \uplus TP \uplus TM$ gives rise to the full set of timed traces (i.e. tA^*), we need only two of the three sets to characterise $\mathcal{P}$.

In a system-environment game play, TF is the set of behaviours that the environment tries to steer the play away from, whereas TM is the set of behaviours that the system tries to steer the play away from. Thus, TF and TM characterise resp. the assumptions AS and guarantees GR of the specification.

Definition 4 (Contract). *A* contract *is a tuple* (I, O, AS, GR)*, where* AS *and* GR *are two disjoint extension-closed trace sets. The contract of* $\mathcal{P}$ *is defined as* $\mathcal{TT}(\mathcal{P}) := (I, O, TF, TM)$*.*

We say the contract of a specification $\mathcal{P}$ is *realisable* iff $\overline{GR}$ in $\mathcal{TT}(\mathcal{P})$ is *I-receptive.* A trace set TT is *I-receptive* iff, for each $tt \in TT$, we have (1) $tt ^\frown \langle e \rangle \in TT$ for all $e \in I$ and (2) $tt ^\frown \langle d \rangle \notin TT$ for some $d \in \mathbb{R}^{>0}$ implies there exists some $0 \leq d_0 < d$ and $e_0 \in O$ s.t. $tt ^\frown \langle d_0, e_0 \rangle \in TT$.

We say the contract of a specification $\mathcal{P}$ is *normalisable* iff $\overline{AS}$ in $\mathcal{TT}(\mathcal{P})$ is *O-receptive.* A trace set TT is *O-receptive* iff, for each $tt \in TT$, we have (1) $tt ^\frown \langle e \rangle \in TT$ for all $e \in O$ and (2) $tt ^\frown \langle d \rangle \notin TT$ for some $d \in \mathbb{R}^{>0}$ implies there exists some $0 \leq d_0 < d$ and $e_0 \in I$ s.t. $tt ^\frown \langle d_0, e_0 \rangle \in TT$.

We can lift the realisation and normalisation operations to contracts:

Definition 5 (Realisation). *The realisation of a contract is* $(I, O, AS, GR)^R = (I, O, AS \backslash GR^R, GR^R)$*, where* GR^R *is the least extension-closed superset of* GR *s.t. no* $tt \in \overline{GR^R}$ *is an auto-*$\top$ *or semi-*$\top$*.*

We say a trace $tt \in TT$ is an *auto*-$\top$ iff $tt ^\frown \langle e \rangle \notin TT$ for some $e \in I$. A trace $tt \in TT$ is an *semi*-$\top$ iff there exists some $d \in \mathbb{R}^{>0}$ s.t. $tt ^\frown \langle d \rangle \notin TT$ and $tt ^\frown \langle d_0, e_0 \rangle \notin TT$ for all $0 \leq d_0 < d$ and $e_0 \in O$. It is easy to verify $\overline{GR^R}$ is I-receptive and $\mathcal{TT}(\mathcal{P})^R = \mathcal{TT}(\mathcal{P}^R)$.

Dually, we can define a trace $tt \in TT$ being an *auto*-$\perp$ or *semi*-$\perp$.

Definition 6 (Normalisation). *Given a contract* (I, O, AS, GR)*, we define* $(I, O, AS, GR)^N = (I, O, AS^N, GR \backslash AS^N)$*, where* AS^N *is the least extension-closed superset of* AS *s.t. no* $tt \in \overline{AS^N}$ *is an auto-*$\perp$ *or semi-*$\perp$*.*

It is easy to verify that $\overline{AS^N}$ is O-receptive and $\mathcal{TT}(\mathcal{P})^N = \mathcal{TT}(\mathcal{P}^N)$.

[21] Bertrand Meyer [18] and Ralph Back [4] first coined the terminology of *contract* in the context of programming languages.

The theory of realisable contracts. A realisable specification gives rise to a realisable contract. Over realisable specifications, our contract theory, with the assistance of normalisation operation, provides an alternative characterisation of $\simeq_r$, which says that a realisable specification $\mathcal{P}$ is a refinement of another one $\mathcal{Q}$ iff $\mathcal{P}$ has less assumptions and more guarantees than $\mathcal{Q}$.

Definition 7 (Neutral contract). *A contract (I, O, AS, GR) is* neutral *iff $\overline{AS}$ is O-receptive and $\overline{GR}$ is I-receptive.*

The neutral contract of the above $\mathcal{P}$ is defined as $\mathcal{CT}(\mathcal{P}) := \mathcal{TT}(\mathcal{P})^N$.

Theorem 1. *Given realisable specifications $\mathcal{P}_0$ and $\mathcal{P}_1$ with $\mathcal{CT}(\mathcal{P}_0) = (I, O, AS_0, GR_0)$ and $\mathcal{CT}(\mathcal{P}_1) = (I, O, AS_1, GR_1)$, $\mathcal{P}_0 \sqsubseteq_r \mathcal{P}_1$ iff $AS_1 \subseteq AS_0$ and $GR_0 \subseteq GR_1$.*

Based on neutral contracts, we present the trace semantics of the parallel, disjunction, conjunction and quotient operations. The core part of the operations is based on a set of patterns originally presented in [20]. The specialisation required for the timed theory lies in the application of closure conditions like normalisation, realisation and *alphabet enlargement*.

Alphabet enlargement. Given a set Δ of actions disjoint from $I \cup O$, we define $(I, O, AS, GR)^{\Delta} := (I \cup \Delta, O, AS^{\Delta}, GR^{\Delta})$, where $TT^{\Delta} := \{tt : (tA \cup \Delta)^* \mid tt \upharpoonright tA \in TT\} \cdot (tA \cup \Delta)^*$.

In the rest of the section we consider two realisable specifications $\mathcal{P}_i$ for $i \in \{0, 1\}$ with $\mathcal{CT}(\mathcal{P}_i) = (I_i, O_i, AS_i, GR_i)$ and $\overline{i} = 1 - i$.

Proposition 2 (Parallel Composition). *If realisable specifications $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\|$-composable, then $\mathcal{CT}(\mathcal{P}_0 \parallel \mathcal{P}_1) = (I, O, (AS_0^{\Delta_0} \cup AS_1^{\Delta_1}) \backslash (GR_0^{\Delta_0} \cup GR_1^{\Delta_1}), GR_0^{\Delta_0} \cup GR_1^{\Delta_1})^N$, where $I = (I_0 \cup I_1) \backslash O$, $O = O_0 \cup O_1$, $\Delta_0 = A_1 \backslash A_0$ and $\Delta_1 = A_0 \backslash A_1$.*

Intuitively, the above says that the composed guarantees are the union of component guarantees, whilst the composed assumptions are the union of component assumptions *with those fulfilled by the composed guarantees deducted.*

Proposition 3 (Disjunction). *If realisable specifications $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\vee$-composable, then $\mathcal{CT}(\mathcal{P}_0 \vee \mathcal{P}_1) = (I, O, AS_0 \cup AS_1, GR_0 \cap GR_1)^N$, where $I = I_0 = I_1$ and $O = O_0 = O_1$.*

Disjunction places union over assumptions but intersection over guarantees.

Proposition 4 (Conjunction). *If realisable specifications $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\wedge$-composable, then $\mathcal{CT}(\mathcal{P}_0 \wedge \mathcal{P}_1) = (I, O, AS_0 \cap AS_1, GR_0 \cup GR_1)^R$, where $I = I_0 = I_1$ and $O = O_0 = O_1$.*

Conjunction places union over guarantees but intersection over assumptions.

Proposition 5 (Quotient). *If realisable specifications $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\%$-composable, then $\mathcal{CT}(\mathcal{P}_0 \% \mathcal{P}_1) = (I, O, AS_0 \cup GR_1^{\Delta_1}, (GR_0 \backslash GR_1^{\Delta_1}) \cup (AS_1^{\Delta_1} \backslash AS_0))^R$, where $I = I_0 \cup O_1$, $O = O_0 \backslash O_1$ and $\Delta_1 = A_0 \backslash A_1$.*

The composed assumptions of quotient is the union of $\mathcal{P}_0$-assumptions and $\mathcal{P}_1$-guarantees, whilst the composed guarantees is the union of (1) $\mathcal{P}_0$-guarantees outside of $\mathcal{P}_1$-guarantees and (2) $\mathcal{P}_1$-assumptions outside of $\mathcal{P}_0$-assumptions.

Mirror. The operation simply interchanges assumptions and guarantees.

Proposition 6. *$\mathcal{CT}(\mathcal{P}^{\neg}) = (O, I, GR, AS)$.*

Based on the above theorem we can prove the congruence result.

Theorem 2. *$\simeq_r$ is a congruence w.r.t. $\|$, $\vee$, $\wedge$ and $\%$, subject to composability.*

8 Comparison with Related Work

Our framework builds on the timed specification theories of [12,13] and [10], albeit with significant differences.

Formalism. All three theories are based on variants of Timed I/O Automata. Our variant, like that of [12,13], uses two invariants (aka *input/output invariants* in [12,13]) in order to recover the duality between assumptions and guarantees; whereas the TIOAs in [10] possess no such duality. Our TIOA semantics, on the other hand, differs from those of [12,13] in the formulation of timed games and adoption of $\top/\bot$, which enable us to reduce the two transition relations in [12,13] to the one relation of Sect. 2.

Timed Game. Both [12,13] and [10] use two-player games, whereas our theory uses a three-player game (with a coin), which is crucial for uncovering the interference between the dual pair of two-player games, normalisation and realisation.

Even with the reduction to two-player games, our treatment of timed games is still different. In comparison with [12,13], our games require that *in each move a finite delay is followed by an action.* Therefore, a play cannot have consecutive delay moves and the possibility of zeno plays (i.e. an infinite play generating a finite trace) is ruled out. Furthermore, finite plays ending in timestop or timelock (i.e. semi-$\top$) can also be removed since we have the realisation game.

In comparison with [10], which is based on the timed game framework of [3,7], our games are strictly more aggressive in classifying winning states. For instance, [3,7] do not classify auto-$\top/\bot$ as winning states.

Linking with refinement calculus. The introduction of $\top$ and $\bot$, inspired by *abort* and *magic* of refinement calculus, significantly simplifies our theory (esp. the operator and refinement-relation definitions and the duality of games), in addition to pointing towards future theory unification.

In contrast, without $\top$ and $\bot$ the pursuit of duality in [12,13] does not end with a simplified theory[22]; especially it misses the second game in duality.

[22] [12,13] focuses on the definition of one operator, parallel composition, which is of considerable complexity.

On the other hand, [10] makes no attempt to link with refinement calculus.

Linear-time and Non-determinism. [10] and [11–13] uses timed alternating simulation as refinement, which (1) does not admit the weakest precongruence and (2) restricts [10,12,13] to consider only deterministic timed systems.

In contrast, we use linear-time semantics that gives rise to both the weakest precongruence and a $\top/\bot$-sensitive determinisation procedure, enabling us to handle non-deterministic timed systems.

Untimed theories. Finally, we remark that our linear-time specification theory owes much to the pioneering work on trace theories for asynchronous circuit verification, especially Dill's trace theory [14]. It is from this community that we take inspiration for the notion of game synthesis, double-trace semantics, auto-$\bot$ (aka auto-failure) and the derivation of quotient from mirror.[23]

In comparison with untimed theories, where only one game with auto-$\bot$ is required,[24] the timed theory requires timestop, two games *in duality*, three players and the new notion of semi-$\top/\bot$. Furthermore, with the use of invariants and co-invariants in timed specifications, timed theory can give a systematic treatment to liveness based on finite traces.

9 Conclusion and Future Work

We have devised a fully compositional specification theory for realisable real-time components. The linear-time theory enjoys strong algebraic properties, supports a full set of composition operators, and admits the weakest substitutive precongruence preserving safety and bounded-liveness error freedom.

Acknowledgments. We benefit from discussions with Prof. David Dill and Prof. Jeff Sanders on *timed extension of trace theory* and *refinement calculus*.

References

AQ3

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**, 183–235 (1994)
2. Armstrong, P.J., Lowe, G., Ouaknine, J., Roscoe, A.W.: Model checking timed CSP. In: A Festschrift on the Occasion of H. Barringer's 60th Birthday (2014)
3. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: IFAC Symposium on System Structure and Control, Elsevier (1998)
4. Ralph-Johan, B., Wright, J.: Refinement Calculus: A Systematic Introduction. Springer, New York (1998)
5. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T., Larsen, K.: Contracts for systems design. Technical report RR-8147, S4 team, INRIA, November 2012

[23] The mirror-based definition of quotient (for the untimed case) was first presented by Verhoeff as his Factorisation Theorem [24].

[24] Composition of untimed specifications will not generated new unrealisable behaviours.

6. Bertrand, N., Stainer, A., Jéron, T., Krichen, M.: A game approach to determinize timed automata. In: Hofmann, M. (ed.) FoSSaCS 2011. LNCS, vol. 6604, pp. 245–259. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19805-2_17
7. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005). doi:10.1007/11539452_9
8. Chilton, C., Jonsson, B., Kwiatkowska, M.: Compositional assume-guarantee reasoning for input, output component theories. Sci. Comput. Program. **91**, 115–137 (2014). Part A
9. Chilton, C., Kwiatkowska, M., Wang, X.: Revisiting timed specification theories: a linear-time perspective. In: FORMATS 2012 (2012)
10. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: HSCC 2010 (2010)
11. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC/FSE 2001 (2001)
12. de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed interfaces. In: EMSOFT 2002, vol. 2491, pp. 108–122 (2002)
13. de Alfaro, L., Stoelinga, M.: Interfaces: a game-theoretic framework for reasoning about component-based systems. Electron. Notes Theoret. Comput. Sci. **97**, 3–23 (2004)
14. Dill, D.L.: Trace theory for automatic hierarchical verification of speed-independent circuits. In: ACM Distinguished Dissertations. MIT Press (1989)
15. Ebergen, J.C.: A technique to design delay-insensitive VLSI circuits. Technical report CS-R8622, Centrum voor Wiskunde en Informatica, June 1986
16. He, J., Hoare, C.A.R., Sanders, J.W.: Data refinement refined resume. In: Robinet, B., Wilhelm, R. (eds.) ESOP 1986. LNCS, vol. 213, pp. 187–196. Springer, Heidelberg (1986). doi:10.1007/3-540-16442-1_14
17. Hoare, C.A.R., He, J., Sanders, J.W.: Prespecification in data refinement. Inf. Process. Lett. **25**(2), 71–76 (1987)
18. Meyer, B.: Design by contract. In: Advances in Object-Oriented Software Engineering. Prentice Hall (1991)
19. Morgan, C.C.: Programming from Specifications. Prentice Hall International Series in Computer Science, 2nd edn. Prentice Hall, Upper Saddle River (1994)
20. Negulescu, R.: Process spaces. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 199–213. Springer, Heidelberg (2000). doi:10.1007/3-540-44618-4_16
21. Reed, G.M., Roscoe, A.W., Schneider, S.A.: CSP and Timewise Refinement, pp. 258–280. Springer, London (1991)
22. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall, Upper Saddle River (1998)
23. van de Snepscheut, J.L.A.: Trace Theory and VLSJ Design. Lecture Notes in Computer Science, vol. 200. Springer, Heidelberg (1985)
24. Verhoeff, T.: A Theory of Delay-Insensitive Systems. Ph.D. thesis, Dept. of Math. and C.S., Eindhoven University of Technology, May 1994
25. Wang, X.: Maximal confluent processes. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 188–207. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31131-4_11
26. Wang, X., Kwiatkowska, M.Z.: On process-algebraic verification of asynchronous circuits. Fundam. Inform. **80**(1–3), 283–310 (2007)

Author Queries

Chapter 2

Query Refs.	**Details Required**	**Author's response**
AQ1	Please confirm if the corresponding author and his mail-id is correctly identified. Amend if necessary.	
AQ2	Per Springer style, both city and country names must be present in the affiliations. Accordingly, we have inserted the city names "Oxford and Sketty" in affiliations "1 and 2". Please check and confirm if the inserted city names are correct. If not, please provide us with the correct city names.	
AQ3	Please check and confirm the edit made in Refs. [13, 15 and 16]	

MARKED PROOF

Please correct and return this set

Please use the proof correction marks shown below for all alterations and corrections. If you wish to return your proof by fax you should ensure that all amendments are written clearly in dark ink and are made well within the page margins.

Instruction to printer	*Textual mark*	*Marginal mark*
Leave unchanged	• • • under matter to remain	✓⃝
Insert in text the matter indicated in the margin	⋏	New matter followed by ⋏ or ⋏ⓧ2
Delete	/ through single character, rule or underline or ⊢⊣ through all characters to be deleted	∂/ or ∂/ⓧ2
Substitute character or substitute part of one or more word(s)	/ through letter or ⊢⊣ through characters	new character / or new characters /
Change to italics	— under matter to be changed	⌣
Change to capitals	≡ under matter to be changed	≡
Change to small capitals	= under matter to be changed	=
Change to bold type	∼ under matter to be changed	∼
Change to bold italic	≂ under matter to be changed	≂
Change to lower case	Encircle matter to be changed	ǂ
Change italic to upright type	(As above)	⊹
Change bold to non-bold type	(As above)	⊹
Insert 'superior' character	/ through character or ⋏ where required	Y or ⅄ under character e.g. $\overset{2}{Y}$ or $\overset{2}{⅄}$
Insert 'inferior' character	(As above)	⋏ over character e.g. $\underset{2}{⋏}$
Insert full stop	(As above)	⊙
Insert comma	(As above)	,
Insert single quotation marks	(As above)	$\overset{‘}{Y}$ or $\overset{‘}{⅄}$ and/or $\overset{’}{Y}$ or $\overset{’}{⅄}$
Insert double quotation marks	(As above)	$\overset{“}{Y}$ or $\overset{“}{⅄}$ and/or $\overset{”}{Y}$ or $\overset{”}{⅄}$
Insert hyphen	(As above)	⊢⊣
Start new paragraph	⎍	⎍
No new paragraph	↩	↩
Transpose	⊔⊓	⊔⊓
Close up	linking ⁀‿ characters	⁀‿
Insert or substitute space between characters or words	/ through character or ⋏ where required	Y
Reduce space between characters or words	| between characters or words affected	⊤