# Revisiting Timed Specification Theories: A Linear-Time Perspective

Chris Chilton, Marta Kwiatkowska, and Xu Wang

Department of Computer Science, University of Oxford, UK

**Abstract.** We consider the setting of component-based design for real-time systems with critical timing constraints. Based on our earlier work, we propose a compositional specification theory for timed automata with I/O distinction, which supports substitutive refinement. Our theory provides the operations of parallel composition for composing components at run-time, logical conjunction/disjunction for independent development, and quotient for incremental synthesis. The key novelty of our timed theory lies in a weakest congruence preserving safety as well as bounded liveness properties. We show that the congruence can be characterised by two linear-time semantics, *timed-traces* and *timed-strategies*, the latter of which is derived from a game-based interpretation of timed interaction.

## 1 Introduction

Component-based design methodologies can be encapsulated in the form of compositional specification theories, which allow the mixing of specifications and implementations, admit substitutive refinement to facilitate reuse, and provide a rich collection of operators. Several such theories have been introduced in the literature, but none simultaneously address the following requirements: support for asynchronous input/output (I/O) communication with non-blocking outputs and non-input receptiveness; linear-time refinement preorder, so as to interface with automata and learning techniques; substitutivity of refinement, to allow for component reuse at runtime without introducing errors; and strong algebraic and compositionality properties, to enable offline as well as runtime reasoning.

Previously [1], we developed a linear-time specification theory for reasoning about untimed components that interact by synchronisation of I/O actions. Models can be specified operationally by means of transition systems augmented by an inconsistency predicate on states, or declaratively using traces. The theory admits non-determinism, a substitutive refinement preorder based on traces, and the operations of parallel composition, conjunction and quotient. The refinement is strictly weaker than alternating simulation and is actually the weakest precongruence preserving freeness of inconsistent states.

In this paper we target component-based development for real-time systems with critical timing constraints, such as embedded system components, the middleware layer and asynchronous hardware. Amongst notable works in the literature, we surveyed the theory of timed interfaces [2] and the theory of timed

M. Jurdziński and D. Ničković (Eds.): FORMATS 2012, LNCS 7595, pp. 75–90, 2012.

specifications [3]. Though both support I/O distinctions, their refinement relations are not linear time: in [2], refinement (compatibility) is based on timed games, and in [3] it is a timed version of the alternating simulation originally defined for interface automata [4]. Consequently, it is too strong for determining when a component can be safely substituted for another. As an example, consider the transition systems $P$ and $Q$ in Figure 3: these should be equivalent in the sense of substitutivity under any environment, and are equivalent in our formulation (Definition 5), but they are not so according to timed alternating simulation.

*Contributions.* We formulate an elegant timed, asynchronous specification theory based on finite traces which supports substitutive refinement, as a timed extension of the linear-time specification theory of [1]. We allow for both operational descriptions of components, as well as declarative specifications based on traces. Our operational models are a variant of timed automata with I/O distinction (although we do not insist on input-enabledness, cf [5]), augmented by two special states: the *inconsistent* state $\bot$ represents safety and boundedliveness errors, while the *timestop* state $\top$ is a novel addition representing either unrealisable output (if the component is not willing to produce that output) or unrealisable time-delay (if the delay would violate the invariant on that state).

Timestop models the ability to stop the clock and has been used before in embedded system and circuit design [6,7]. It is notationally convenient, accounting for simpler definitions and a cleaner formalism. By enhancing the automata with the notion of *co-invariant*, we can, for the first time, distinguish the roles of input/output guards and invariant/co-invariants as specifying safety and boundedliveness timed assumptions/guarantees. We emphasise that this is achieved with finite traces only; note that in the untimed case it would be necessary to extend to infinite traces to model liveness. In addition to *timed-trace semantics*, we present *timed-strategy semantics*, which coincides with the former but relates our work closer to the timed-game frameworks used by [3] and [2], and could in future serve as a guide to implementation of the theory. Finally, the *substitutive refinement* of our framework gives rise to the weakest congruence preserving $\bot$-freeness, which is not the case in the formalism of [3].

*Related work.* Our work can be seen as an alternative to the timed theories of [2,3]. Being linear-time in spirit, it is also a generalisation of [8], an untimed theory inspired by asynchronous circuits, and Dill's trace theory [9]. The specification theory in [3] also introduces parallel, conjunction and quotient, but uses timed alternating simulation as refinement, which does not admit the weakest precongruence. An advantage of [3] is the algorithmic efficiency of branching-time simulation checking as well as the implementation reported in [10]. We briefly mention other related works, which include timed modal transition systems [11,12], the timed I/O model [5,13] and asynchronous circuits and embedded systems [14,15]. A more detailed comparison based on the technical details of our work is included in Section 5. A full version of this paper including an even greater comparison with related work, in addition to proofs, is available as [16].

## 2 Formal Framework

In this section we introduce timed I/O automata, timed I/O transition systems and a semantic mapping from the former to the latter. Timed I/O automata are compact representations of timed I/O transition systems. We also present an operational specification theory based on timed I/O transition systems, which are endowed with a richer repertoire of semantic machinery than the automata.

### 2.1 Timed I/O Automata

*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 $x \bowtie d$ and $x - y \bowtie d$ where $x, y \in X$, $\bowtie \in \{\leq, <, =, >, \geq\}$, and $d \in \mathbb{N}$.

A *clock valuation* over $X$ is a map $t$ that assigns to each clock variable $x$ in $X$ a real value from $\mathbb{R}^{\geq 0}$. We say $t$ satisfies $cc$, written $t \in cc$, if $cc$ evaluates to true under valuation $t$. $t + d$ denotes the valuation derived from $t$ by increasing the assigned value on each clock variable by $d \in \mathbb{R}^{\geq 0}$ time units. $t[rs \mapsto 0]$ denotes the valuation obtained from $t$ by resetting the clock variables in $rs$ to 0. Sometimes we use 0 for the clock valuation that maps all clock variables to 0.

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

- $C \subseteq X$ *is a finite set of clock variables*
- $A\ (= I \uplus O)$ *is a finite alphabet, consisting of inputs* $I$ *and outputs* $O$
- $L$ *is a finite set of* locations *and* $l^0 \in L$ *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.*

We use $l, l', l_i$ to range over $L$ and 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 timed automata that distinguish input from output and invariant from co-invariant. They are similar to existing variants of timed automata with input/output distinction, except for the introduction of co-invariants and non-insistence on input-enabledness. While invariants specify the bounds beyond which time may not progress, co-invariants specify the bounds beyond which the system will *time-out* and enter error states. It is designed for the assume/guarantee specification of timed components, in order to specify both the assumptions made by the component on the inputs and the guarantees provided by the component on the outputs, with respect to timing constraints.

Guards on output transitions express *safety timing guarantees*, while guards on input transitions express *safety timing assumptions*. On the other hand, invariants (urgency) express *liveness timing guarantees* on the outputs at the locations they decorate, while co-invariants (time-out) express *liveness timing assumptions* on the inputs at those locations.

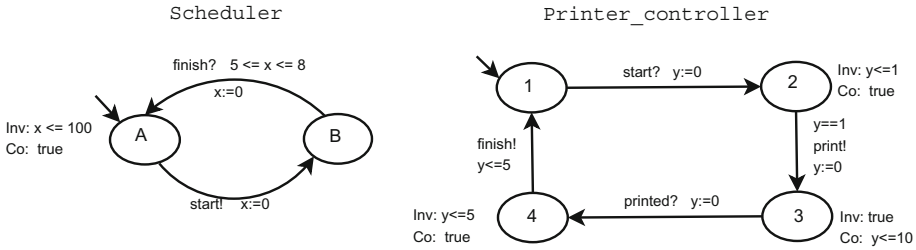


**Fig. 1.** Job scheduler and printer controller

When two components are composed, the parallel composition automatically checks whether the guarantees provided by one component meet the assumptions required by the other. For instance, the unexpected arrival of an input at a particular location and time (indicated by a non-enabled transition) leads to a *safety error* in the parallel composition. The non-arrival of an expected input at a location before its time-out (specified by the co-invariant) leads to a *bounded-liveness error* in the parallel composition.

*Example.* Figure 1 depicts TIOAs representing 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 $t = 100$, the *start* action must be fired within the range $0 \leq t \leq 100$. After *start* has been fired, the clock $x$ is reset to 0 and the scheduler waits (possibly indefinitely) for the job to *finish*. If the job does finish, the scheduler is only willing for this to take place between $5 \leq t \leq 8$ after the job started (safety assumption), otherwise an unexpected input error will be thrown.

The controller waits for the job to *start*, after which it will wait exactly 1 time unit before issuing *print* (forced by the invariant $y \leq 1$ on state 2 and the guard $y = 1$). The controller now requires the printer to indicate the job is *printed* within 10 time units of being sent to the printer, otherwise a time-out error on inputs will occur (co-invariant $y \leq 10$ in state 3 as liveness assumption). After the job has finished printing, the controller must indicate to the scheduler that the job has *finish*ed within 5 time units.

*Notation.* For a set of input actions $I$ and a set of output actions $O$, define $tA = I \uplus O \uplus \mathbb{R}^{>0}$ to be the set of *timed actions*, $tI = I \uplus \mathbb{R}^{>0}$ to be the set of *timed inputs*, and $tO = O \uplus \mathbb{R}^{>0}$ to be the set of *timed outputs*. We use symbols like $\alpha$, $\beta$, etc. to range over $tA$.

A *timed word* (ranged over by $w, w', w_i$ 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 word denoting the observation that action $a$ occurs at 0.33 time units, then another 1.41 time units lapse before the simultaneous occurrence of $b$ and $c$, which is followed by 3.1415 time units of no event occurrence. $\epsilon$ denotes the empty word.

Concatenation of timed words $w$ and $w'$ is obtained by appending $w'$ onto the end of $w$ and coalescing adjacent reals (summing them). Prefix/extension

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

### 2.2 Semantics as Timed I/O Transition Systems

The semantics of TIOAs are given as timed I/O transition systems, which are a special class of infinite labelled transition systems.

**Definition 2.** *A* timed I/O transition system *(TIOTS) is a tuple* $\mathcal{P} = \langle I, O, S, s^0, \rightarrow \rangle$*, where* $I$ *and* $O$ *are the input and output actions respectively,* $S = (L \times \mathbb{R}^C) \uplus \{\bot, \top\}$ *is a set of states,* $s^0 \in S$ *is the designated initial state, and* $\rightarrow \subseteq S \times (I \uplus O \uplus \mathbb{R}^{>0}) \times S$ *is the action and time-labelled transition relation.*

The states of the TIOTS for a TIOA capture the configuration of the automaton, i.e. its location and clock valuation. Therefore, each state of the TIOTS is a pair drawn from $L \times \mathbb{R}^C$, which we refer to as the set of *plain states*. In addition, we introduce two special states $\bot$ and $\top$, which are required for the semantic mapping of disabled inputs/outputs, invariants and co-invariants. In the rest of the paper, we use $p, p', p_i$ to range over $P = L \times \mathbb{R}^C$ while $s, s', s_i$ range over $S$.

$\bot$ is the so-called *inconsistent state*, arising through assumption/guarantee mismatches, i.e. safety and bounded-liveness errors. $\top$ is the so-called *timestop state*, representing the *magic moment* from which time stops elapsing and no error can occur. We assume that $\top$ refines plain states, which in turn refine $\bot$. For technical convenience (e.g. ease of defining time additivity and trace semantics), we require that $\top$ and $\bot$ are a *chaotic states*, i.e. states having self-loops for each $\alpha \in tA$.

On TIOTSs, a disabled input in a state $p$ is equated to an input transition from $p$ to $\bot$, while a disabled output/delay in $p$ is equated to an output/delay from $p$ to $\top$. The intuition here comes from the I/O game perspective. The component controls output and delay, while the environment controls input. $\bot$ is the losing state for the environment, so an input transition from $p$ to $\bot$ is a transition that the environment tries to avoid at all cost (unless there is no choice). $\top$ is the losing state for the component, so an output/delay transition from $p$ to $\top$ is a transition that the component tries to avoid at any cost. Thus we can have two semantic-preserving transformations on TIOTSs.

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 $p$.[1] The $\top$-*completion*, denoted $\mathcal{P}^\top$, adds an $\alpha$-labelled transition from $p$ to $\top$ for every $p \in P$ and $\alpha \in tO$ s.t. $\alpha$ is not enabled at $p$.

Now, the transition relation $\rightarrow$ of the TIOTS is derived from the execution semantics of the TIOA.

**Definition 3.** *Let* $\mathcal{P}$ *be a TIOA. The execution semantics of* $\mathcal{P}$ *is a TIOTS* $\langle I, O, S, s^0, \rightarrow \rangle$*, where:*

[1] $\bot$-completion will make a TIOTS *input-receptive*, i.e. input-enabled in all states.

- $S = (L \times \mathbb{R}^C) \uplus \{\bot, \top\}$
- $s^0 = \top$ *providing* $0 \notin Inv(l^0)$, $s^0 = \bot$ *providing* $0 \in Inv(l^0) \wedge \neg coInv(l^0)$ *and* $s^0 = (l^0, 0)$ *providing* $0 \in Inv(l^0) \wedge coInv(l^0)$,
- $\rightarrow$ *is the smallest relation satisfying:*
  1. *If* $l \xrightarrow{g,a,rs} l'$, $t' = t[rs \mapsto 0]$, $t \in Inv(l) \wedge coInv(l) \wedge g$, *then:*
     *(a)* plain action: $(l, t) \xrightarrow{a} (l', t')$ *providing* $t' \in Inv(l') \wedge coInv(l')$
     *(b)* error action: $(l, t) \xrightarrow{a} \bot$ *providing* $t' \in Inv(l') \wedge \neg coInv(l')$
     *(c)* magic action: $(l, t) \xrightarrow{a} \top$ *providing* $t' \in \neg Inv(l')$ *and* $a \in I$.
  2. plain delay: $(l, t) \xrightarrow{d} (l, t + d)$ *if* $t, t + d \in Inv(l) \wedge coInv(l)$
  3. time-out delay: $(l, t) \xrightarrow{d} \bot$ *if* $t \in Inv(l) \wedge coInv(l)$, $t + d \notin coInv(l)$ *and* $\exists\, 0 < \delta \leq d : t + \delta \in Inv(l) \wedge \neg coInv(l)$.

Note that our semantics tries to minimise the use of transitions leading to $\top/\bot$ states. Thus there are no delay or output transitions leading to $\top$. However, there are *implicit timestops*, which we capture using the concept of *semi-timestop* (i.e. semi-$\top$). We say a plain state $p$ is a *semi*-$\top$ iff 1) all output transitions enabled in $p$ and all of its time-passing successors lead to the $\top$ state, and 2) there exists $d \in \mathbb{R}^{>0}$ s.t. $p \xrightarrow{d} \top$ or $d$ is not enabled in $p$. Thus a semi-$\top$ is a state in which it is impossible for the component to avoid the timestop without suitable inputs from the environment.

The introduction of timestop ($\top$), which can model the operation of stopping the system clock, is an unconventional aspect of our semantics. Certain real-world systems have an inherent ability to stop the clock, e.g. [6,7], which are related to embedded systems and circuit design. When the suspension of clocks is not meaningful, it is necessary to remove timestop in order to leave the so-called realisable behaviour. Timestop is useful even for timestop free systems, as it can significantly simplify operations, such as quotient and conjunction.

*TIOTS terminology.* We say a TIOTS is *deterministic* iff $s \xrightarrow{\alpha} s' \wedge s \xrightarrow{\alpha} s''$ implies $s' = s''$, and 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 the sequel, we only consider time-additive TIOTSs.

Given a TIOTS $\mathcal{P}$, a timed word can be derived from a finite execution of $\mathcal{P}$ by extracting the labels in each transition and coalescing adjacent reals. The timed words derived from such executions are called *traces* of $\mathcal{P}$. We use $tt$, $tt'$, $tt_i$ to range over traces and write $s^0 \stackrel{tt}{\Rightarrow} s$ to denote a finite execution producing $tt$ and leading to $s$.

### 2.3 Operational Specification Theory

In this section we develop a compositional specification theory for TIOTSs based on the operations of parallel composition $\|$, conjunction $\wedge$, disjunction $\vee$ and quotient $\%$. The operators are defined via transition rules that are a variant on synchronised product.

Parallel composition yields a TIOTS that represents the combined effect of its operands interacting with one another. The remaining operations must

**Table 1.** State representations under composition operators

| $\parallel$ | $\top$ | $p_0$ | $\bot$ |
|---|---|---|---|
| $\top$ | $\top$ | $\top$ | $\top$ |
| $p_1$ | $\top$ | $p_0 \times p_1$ | $\bot$ |
| $\bot$ | $\top$ | $\bot$ | $\bot$ |

| $\wedge$ | $\top$ | $p_0$ | $\bot$ |
|---|---|---|---|
| $\top$ | $\top$ | $\top$ | $\top$ |
| $p_1$ | $\top$ | $p_0 \times p_1$ | $p_1$ |
| $\bot$ | $\top$ | $p_0$ | $\bot$ |

| $\vee$ | $\top$ | $p_0$ | $\bot$ |
|---|---|---|---|
| $\top$ | $\top$ | $p_0$ | $\bot$ |
| $p_1$ | $p_1$ | $p_0 \times p_1$ | $\bot$ |
| $\bot$ | $\bot$ | $\bot$ | $\bot$ |

| $\%$ | $\top$ | $p_0$ | $\bot$ |
|---|---|---|---|
| $\top$ | $\bot$ | $\bot$ | $\bot$ |
| $p_1$ | $\top$ | $p_0 \times p_1$ | $\bot$ |
| $\bot$ | $\top$ | $\top$ | $\bot$ |

be explained with respect to a refinement relation, which corresponds to safe-substitutivity in our theory. A TIOTS is a refinement of another if it will work in any environment that the original worked in without introducing safety or bounded-liveness errors. Conjunction yields the coarsest TIOTS that is a refinement of its operands, while disjunction yields the finest TIOTS that is refined by both of its operands. The operators are thus equivalent to the join and meet operations on TIOTSs[2]. Quotient is the adjoint of parallel composition, meaning that $\mathcal{P}_0 \% \mathcal{P}_1$ is the coarsest TIOTS such that $(\mathcal{P}_0 \% \mathcal{P}_1) \| \mathcal{P}_1$ is a refinement of $\mathcal{P}_0$.

Let $\mathcal{P}_i = \langle I_i, O_i, S_i, s_i^0, \rightarrow_i \rangle$ for $i \in \{0, 1\}$ be two TIOTSs that are both $\bot$ and $\top$-completed, satisfying (wlog) $S_0 \cap S_1 = \{\bot, \top\}$. The composition of $\mathcal{P}_0$ and $\mathcal{P}_1$ under the operation $\otimes \in \{\|, \wedge, \vee, \%\}$, written $\mathcal{P}_0 \otimes \mathcal{P}_1$, is only defined when certain *composability* restrictions are imposed on the alphabets of the TIOTSs. $\mathcal{P}_0 \parallel \mathcal{P}_1$ is only defined when the output sets of $\mathcal{P}_0$ and $\mathcal{P}_1$ are disjoint, because an output should be controlled by at most one component. Conjunction and disjunction are only defined when the TIOTSs have *identical alphabets* (i.e. $O_0 = O_1$ and $I_0 = I_1$). This restriction can be relaxed at the expense of more cumbersome notation, which is why we focus on the simpler case in this paper. For the quotient, we require that the alphabet of $\mathcal{P}_0$ *dominates* that of $\mathcal{P}_1$ (i.e. $A_1 \subseteq A_0$ and $O_1 \subseteq O_0$), in addition to $\mathcal{P}_1$ being a deterministic TIOTS. As quotient is a synthesis operator, it is difficult to give a definition using just *state-local* transition rules, since quotient needs global information about the transition systems. This is why we insist on $\mathcal{P}_1$ being deterministic[3].

**Definition 4.** *Let $\mathcal{P}_0$ and $\mathcal{P}_1$ be TIOTSs composable under $\otimes \in \{\|, \wedge, \vee, \%\}$. Then $\mathcal{P}_0 \otimes \mathcal{P}_1 = \langle I, O, S, s^0, \rightarrow \rangle$ is the TIOTS where:*

- *If $\otimes = \|$, then $I = (I_0 \cup I_1) \setminus O$ and $O = O_0 \cup O_1$*
- *If $\otimes \in \{\wedge, \vee\}$, then $I = I_0 = I_1$ and $O = O_0 = O_1$*
- *If $\otimes = \%$, then $I = I_0 \cup O_1$ and $O = O_0 \setminus O_1$*
- *$S = (P_0 \times P_1) \uplus P_0 \uplus P_1 \uplus \{\top, \bot\}$*
- *$s^0 = s_0^0 \otimes s_1^0$*
- *$\rightarrow$ is the smallest relation containing $\rightarrow_0 \cup \rightarrow_1$, and satisfying the rules:*

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

[2] As we write $A \sqsubseteq B$ to mean $A$ is refined by $B$, our operators $\wedge$ and $\vee$ are reversed in comparison to the standard symbols for meet and join.

[3] Technically speaking, the problem is a consequence of state quotient being right-distributive but not left-distributive over state disjunction (cf Table 1).

*We adopt the notation of $s_0 \otimes s_1$ for states, where the associated interpretation is supplied in Table 1. Furthermore, given two plain states $p_i = (l_i, t_i)$ for $i \in \{0, 1\}$, we define $p_0 \times p_1 = ((l_0, l_1), t_0 \uplus t_1)$.*

Table 1 tells us how states should be combined under the composition operators. For parallel, a state is magic if one component state is magic, and a state is error if one component is error while the other is not magic. For conjunction, encountering error in one component implies the component can be discarded and the rest of the composition behaves like the other component. The conjunction table follows the intuition of the join operation on the refinement preorder. Similarly for disjunction. Quotient is the adjoint of parallel composition. If the second component state does not refine the first, the quotient will try to rescue the refinement by producing $\top$ (so that its composition with the second will refine the first). If the second component state does refine the first, the quotient will produce the least refined value so that its composition with the second will not break the refinement.

An *environment* for a TIOTS $\mathcal{P}$ is any TIOTS $\mathcal{Q}$ such that the alphabet of $\mathcal{Q}$ is *complementary* to that of $\mathcal{P}$, meaning $I_\mathcal{P} = O_\mathcal{Q}$ and $O_\mathcal{P} = I_\mathcal{Q}$. Refinement in our framework corresponds to contextual substitutability, in which the context is an arbitrary environment.

**Definition 5.** *Let $\mathcal{P}_{imp}$ and $\mathcal{P}_{spec}$ be TIOTSs with identical alphabets. $\mathcal{P}_{imp}$* refines *$\mathcal{P}_{spec}$, denoted $\mathcal{P}_{spec} \sqsubseteq \mathcal{P}_{imp}$, iff for all environments $\mathcal{Q}$, $\mathcal{P}_{spec} \parallel \mathcal{Q}$ is $\bot$-free implies $\mathcal{P}_{imp} \parallel \mathcal{Q}$ is $\bot$-free. We say $\mathcal{P}_{imp}$ and $\mathcal{P}_{spec}$ are* substitutively equivalent*, i.e. $\mathcal{P}_{spec} \simeq \mathcal{P}_{imp}$, iff $\mathcal{P}_{imp} \sqsubseteq \mathcal{P}_{spec}$ and $\mathcal{P}_{spec} \sqsubseteq \mathcal{P}_{imp}$.*

It is obvious that $\simeq$ induces the weakest equivalence on TIOTSs that preserves $\bot$-freeness. In the sequel, we give two concrete characterisations of $\simeq$ and show it to be a congruence w.r.t. the operators of the specification theory.

The operational definition of quotient requires $\mathcal{P}_1$ to be deterministic. For any TIOTS $\mathcal{P}$, a semantically-equivalent deterministic component can be obtained, denoted $\mathcal{P}^D$, by means of a modified subset construction acting on $(\mathcal{P}^\bot)^\top$. For any subset $S_0$ of states reachable by a given trace, we only keep those which are minimal w.r.t. the state refinement relation. So if the current state subset $S_0$ contains $\bot$, the procedure reduces $S_0$ to $\bot$; if $\bot \notin S_0 \neq \{\top\}$, it reduces $S_0$ by removing any potential $\top$ in $S_0$.[4]

**Proposition 1.** *For any TIOTS $\mathcal{P}$, it holds that $\mathcal{P} \simeq \mathcal{P}^D$.*

Equipped with determinisation, quotient is a fully defined operator on any pair of TIOTSs. Furthermore, we can give an alternative (although substitutively equivalent) formulation of quotient as the derived operator $(\mathcal{P}_0^\neg \parallel \mathcal{P}_1)^\neg$, where $\neg$ is a mirroring operation that first determinises its argument, then interchanges the input and output sets, as well as the $\top$ and $\bot$ states.

[4] A detailed definition of transforming untimed non-deterministic systems into substitutively-equivalent deterministic ones is contained in Definition 4.2 of [8].

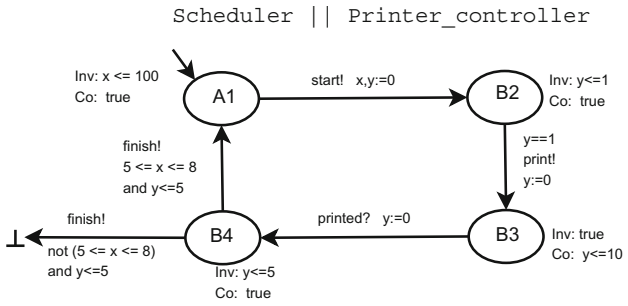


**Fig. 2.** Parallel composition of the job scheduler and printer controller

*Example.* 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 a safety error will occur. This is indicated by the transition to $\bot$ when the guard constraint $5 \leq x \leq 8$ is not satisfied.

## 3 Timed I/O Game

Our specification theory can be seen as an I/O game between a *component* and an *environment* that uses a *coin* to break ties. The specification of a component (in the form of a TIOA or TIOTS) is built to encode the set of strategies possible for the component in the game (just like an NFA encodes a set of words).

- Given two TIOTSs $\mathcal{P}$ and $\mathcal{Q}$ with identical alphabets, we say $\mathcal{P}$ is a partial unfolding [17] of $\mathcal{Q}$ if there exists a function $f$ from $S_\mathcal{P}$ to $S_\mathcal{Q}$ s.t. 1) $f$ maps $\top$ to $\top$, $\bot$ to $\bot$, and plain states to plain states, 2) $f(s^0_\mathcal{P}) = s^0_\mathcal{Q}$, and 3) $p \xrightarrow{\alpha}_\mathcal{P} s \Rightarrow f(p) \xrightarrow{\alpha}_\mathcal{Q} f(s)$.
- 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 $p = p'$.
- We say an acyclic TIOTS is a *simple path* if 1) $p \xrightarrow{a} s' \wedge p \xrightarrow{\alpha} s''$ implies $s' = s''$ and $a = \alpha$ and 2) $p \xrightarrow{d} s' \wedge p \xrightarrow{d} s''$ implies $s' = s''$.
- We say a simple path $\mathcal{L}$ is a *run* of $\mathcal{P}$ if $\mathcal{L}$ is a partial unfolding of $\mathcal{P}$.

*Strategies.* A *strategy* $\mathcal{G}$ is a deterministic tree TIOTS s.t. each plain state in $\mathcal{G}$ is ready to accept all possible inputs by the environment, but allows a single move (delay or output) by the component, i.e. $eb_\mathcal{G}(p) = I \uplus mv_\mathcal{G}(p)$ s.t. $mv_\mathcal{G}(p) = \{a\}$ for some $a \in O$ or $mv_\mathcal{G}(p) \subseteq \mathbb{R}^{>0}$, where $eb_\mathcal{G}(p)$ denotes the set of enabled timed actions in state $p$ of LTS $\mathcal{G}$, and $mv_\mathcal{G}(p)$ denotes the unique component move allowed by $\mathcal{G}$ at $p$.

A TIOTS $\mathcal{P}$ *contains* a strategy $\mathcal{G}$ if $\mathcal{G}$ is a partial unfolding of $(\mathcal{P}^\bot)^\top$. The set of strategies contained in $\mathcal{P}$ is denoted $stg(\mathcal{P})$. Since it makes little sense to

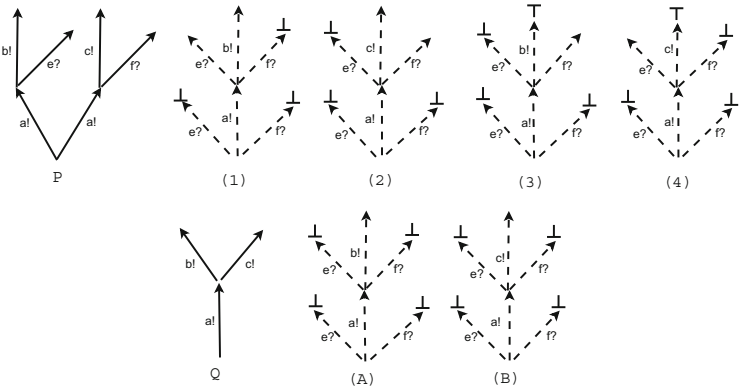


**Fig. 3.** Strategy example

distinguish strategies that are isomorphic, we will freely use strategies to refer to their isomorphism classes and write $\mathcal{G} = \mathcal{G}'$ to mean $\mathcal{G}$ and $\mathcal{G}'$ are isomorphic.

Figure 3 illustrates the idea of strategies. For simplicity, we use two untimed transition systems $\mathcal{P}$ and $\mathcal{Q}$ with identical alphabets $I = \{e, f\}$ and $O = \{a, b, c\}$. The transition systems use solid lines, while strategies use dotted lines. Plain states are unmarked, while the $\top$ and $\bot$ states are labelled as such[5]. A subset of the strategies for $\mathcal{P}$ and $\mathcal{Q}$ are shown on the right hand side of the respective components. Note that strategies 3 and 4 arise through $\top$-completion.

*Comparing strategies.* When the game is played, the component tries to avoid reaching $\top$, while the environment tries to avoid reaching $\bot$. Strategies in $stg(\mathcal{P})$ vary in their effectiveness to achieve this objective, which induces a hierarchy on strategies that closely resemble one another. We say $\mathcal{G}$ and $\mathcal{G}'$ 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, it means $\mathcal{G}$ and $\mathcal{G}'$ propose the same move at the 'same' states. For instance, the strategies 1, 3 and $A$ in Figure 3 are pairwise affine and so are the strategies 2, 4 and $B$.

Given two affine strategies $\mathcal{G}$ and $\mathcal{G}'$, we say $\mathcal{G}$ is *more 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 $stg(\mathcal{P})$, or more generally, over any set of strategies with identical alphabets. For instance, strategy $A$ is more aggressive than 1 and 3, while strategy $B$ is more aggressive than 2 and 4.

When the game is played, the component $\mathcal{P}$ prefers to use the maximally aggressive strategies in $stg(\mathcal{P})$[6]. Thus two components that differ only in non-maximally aggressive strategies should be equated. We define the *strategy semantics* of component $\mathcal{P}$ to be $[\mathcal{P}]_s = \{\mathcal{G}' \mid \exists \mathcal{G} \in stg(\mathcal{P}) : \mathcal{G} \preceq \mathcal{G}'\}$, i.e. the upward-closure of $stg(\mathcal{P})$ w.r.t. $\preceq$.

[5] For simplicity, we allow multiple copies of $\top$ and $\bot$, which are assumed to be chaotic.
[6] This is because our semantics is designed to preserve $\bot$ rather than $\top$.

*Game rules.* When a component strategy $\mathcal{G}$ is played against an environment strategy $\mathcal{G}'$, at each game state (i.e. a product state $p_\mathcal{G} \times p_{\mathcal{G}'}$) $\mathcal{G}$ and $\mathcal{G}'$ each propose a move (i.e. $mv_\mathcal{G}(p_\mathcal{G})$ and $mv_{\mathcal{G}'}(p_{\mathcal{G}'})$). If one of them is a delay and the other is an action, the action will prevail. If both propose delay moves (i.e. $mv_\mathcal{G}(p_\mathcal{G}), mv_{\mathcal{G}'}(p_{\mathcal{G}'}) \subseteq \mathbb{R}^{>0}$), the smaller one (w.r.t. set containment) will prevail.[7]

Since a delay move proposed at a strategy state is the maximal set of possible delays enabled at that state, the next move proposed at the new state after firing the set must be an action move (due to time additivity). Thus a play cannot have two consecutive delay moves.

If, however, both propose action moves, there will be a tie, which will be resolved by tossing the coin. For uniformity's sake, the coin can be treated as a special component. A strategy of the coin is a function $h$ from $tA^*$ to $\{0, 1\}$. We denote the set of all possible coin strategies as $H$.

A play of the game can be formalised as a composition of three strategies, one each from the component, environment and coin, denoted $\mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q}$. At a current game state $p_\mathcal{P} \times p_\mathcal{Q}$, if the prevailing action is $\alpha$ and we have $p_\mathcal{P} \xrightarrow{\alpha} s'_\mathcal{P}$ and $p_\mathcal{Q} \xrightarrow{\alpha} s'_\mathcal{Q}$, then the next game state is $s_\mathcal{P} \parallel s_\mathcal{Q}$. The play will stop when it reaches either $\top$ or $\bot$. The composition will produce a simple path $\mathcal{L}$ that is a run of $\mathcal{P} \parallel \mathcal{Q}$. Since $\mathcal{P} \parallel \mathcal{Q}$ gives rise to a *closed system* (i.e. the input alphabet is empty), a run of $\mathcal{P} \parallel \mathcal{Q}$ is a strategy of $\mathcal{P} \parallel \mathcal{Q}$.

Thus, strategy composition of $\mathcal{P}$ and $\mathcal{Q}$ is closely related to their parallel composition: $stg(\mathcal{P} \parallel \mathcal{Q}) = \{\mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q} \mid \mathcal{G}_\mathcal{P} \in stg(\mathcal{P}), \mathcal{G}_\mathcal{Q} \in stg(\mathcal{Q}) \text{ and } h \in H\}$.

*Parallel composition.* Strategy composition, like component parallel composition, can be generalised to any pair of components $\mathcal{P}$ and $\mathcal{Q}$ with *composable alphabets*. That is, $O_\mathcal{P} \cap O_\mathcal{Q} = \{\}$. For such $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q}$ gives rise to a tree rather than a simple path TIOTS. That is, at each game state $p_\mathcal{P} \times p_\mathcal{Q}$, besides firing the prevailing $\alpha \in tO_\mathcal{P} \cup tO_\mathcal{Q}$, we need also to fire 1) all the synchronised inputs, i.e. $e \in I_\mathcal{P} \cap I_\mathcal{Q}$, and reach the new game state $s_\mathcal{P} \parallel s_\mathcal{Q}$ (assuming $p_\mathcal{P} \xrightarrow{e} s_\mathcal{P}$ and $p_\mathcal{Q} \xrightarrow{e} s_\mathcal{Q}$) and 2) all the independent inputs, i.e. $e \in (I_\mathcal{P} \cup I_\mathcal{Q}) \setminus (A_\mathcal{P} \cap A_\mathcal{Q})$, and reach the new game state $s_\mathcal{P} \times p_\mathcal{Q}$ or $p_\mathcal{P} \times s_\mathcal{Q}$. It is easy to verify that $\mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q}$ is a strategy of $\mathcal{P} \parallel \mathcal{Q}$.

*Conjunction/disjunction. Strategy conjunction* (&) and *strategy disjunction* (+) are binary operators defined only on pairs of affine strategies, by $\mathcal{G}\&\mathcal{G}' = \mathcal{G} \wedge \mathcal{G}'$ and $\mathcal{G} + \mathcal{G}' = \mathcal{G} \vee \mathcal{G}'$. If $\mathcal{G}$ and $\mathcal{G}'$ are not affine, $\mathcal{G} \wedge \mathcal{G}'$ and $\mathcal{G} \vee \mathcal{G}'$ may not produce a strategy. From Figure 3, the disjunction of strategies 1 and 2 will produce a transition system that stops to output after the $a$ transition.

*Refinement.* Equality of strategies induces an equivalence on TIOTSs: $\mathcal{P}$ and $\mathcal{Q}$ are *strategy equivalent* iff $[\mathcal{P}]_s = [\mathcal{Q}]_s$. However, strategy equivalence is too fine for the purpose of *substitutive refinement* (cf Definition 5). For instance,

[7] Note that all invariants and co-invariants are downward-closed. Thus a delay move can be respresented as a time interval from 0 to some $d \in \mathbb{R}^{\geq 0}$.

transition systems $\mathcal{P}$ and $\mathcal{Q}$ in Figure 3 are substitutively equivalent, but are not strategy equivalent, because 1, 2, 3 and 4 are strategies of $\mathcal{Q}$ (due to upward-closure w.r.t. $\preceq$), while $A$ and $B$ are not strategies of $\mathcal{P}$.

However, we demonstrate that *substitutive equivalence is reducible to strategy equivalence* providing we perform *disjunction closure* on strategies.

**Lemma 1.** *Given a pair of affine component strategies $\mathcal{G}_0$ and $\mathcal{G}_1$, $\mathcal{G}_0 \parallel_h \mathcal{G}$ and $\mathcal{G}_1 \parallel_h \mathcal{G}$ are $\perp$-free for a pair of environment and coin strategies $\mathcal{G}$ and $h$ iff $\mathcal{G}_0 + \mathcal{G}_1 \parallel_h \mathcal{G}$ is $\perp$-free.*

We say $\Pi^+$ is a *disjunction closure* of set of strategies $\Pi$ iff it is the least superset of $\Pi$ s.t. $\mathcal{G} + \mathcal{G}' \in \Pi^+$ for all pairs of affine strategies $\mathcal{G}, \mathcal{G}' \in \Pi^+$. It is easy to see disjunction closure preserves upward-closedness of strategy sets.

**Proposition 2.** *Disjunction closure is determinisation: $[\mathcal{P}^D]_s = [\mathcal{P}^D]_s^+ = [\mathcal{P}]_s^+$.*

**Lemma 2.** *For any TIOTS $\mathcal{P}$, $[\mathcal{P}^\neg]_s^+ = \{\mathcal{G}_{\mathcal{P}^\neg} \mid \forall\, \mathcal{G}_\mathcal{P} \in [\mathcal{P}]_s^+, h \in H : \mathcal{G}_{\mathcal{P}^\neg} \parallel_h \mathcal{G}_\mathcal{P}$ is $\perp$-free$\}$.*

**Theorem 1.** *Given TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{P} \sqsubseteq \mathcal{Q}$ iff $[\mathcal{Q}]_s^+ \subseteq [\mathcal{P}]_s^+$.*

Looking at Figure 3, the disjunction of strategies 1 and 3 produces $A$, while the disjunction of strategies 2 and 4 produces $B$. Thus $[\mathcal{P}]_s^+ = [\mathcal{Q}]_s^+$.

*Relating operational composition to strategies.* The operations of parallel composition, conjunction, disjunction and quotient defined on the operational models of TIOTSs (Section 2.3) can be characterised by simple operations on strategies in the game-based setting.

**Lemma 3.** *For $\|$-composable TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $[\mathcal{P} \parallel \mathcal{Q}]_s^+ = \{\mathcal{G}_{\mathcal{P}\|\mathcal{Q}} \mid \exists\, \mathcal{G}_\mathcal{P} \in [\mathcal{P}]_s^+, \mathcal{G}_\mathcal{Q} \in [\mathcal{Q}]_s^+, h \in H : \mathcal{G}_\mathcal{P} \parallel_h \mathcal{G}_\mathcal{Q} \preceq \mathcal{G}_{\mathcal{P}\|\mathcal{Q}}\}$.*

**Lemma 4.** *For $\vee$-composable TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $[\mathcal{P} \vee \mathcal{Q}]_s^+ = ([\mathcal{P}]_s^+ \cup [\mathcal{Q}]_s^+)^+$.*

**Lemma 5.** *For $\wedge$-composable TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $[\mathcal{P} \wedge \mathcal{Q}]_s^+ = [\mathcal{P}]_s^+ \cap [\mathcal{Q}]_s^+$.*

**Lemma 6.** *For $\%$-composable TIOTSs $\mathcal{P}$ and $\mathcal{Q}$, $[\mathcal{P}\%\mathcal{Q}]_s^+ = \{\mathcal{G}_{\mathcal{P}\%\mathcal{Q}} \mid \forall\, \mathcal{G}_\mathcal{Q} \in [\mathcal{Q}]_s^+, h \in H : \mathcal{G}_{\mathcal{P}\%\mathcal{Q}} \parallel_h \mathcal{G}_\mathcal{Q} \in [\mathcal{P}]_s^+\}$.*

Thus, conjunction and disjunction are the join and meet operations, and quotient produces the coarsest TIOTS s.t. $(\mathcal{P}_0 \% \mathcal{P}_1) \| \mathcal{P}_1$ is a refinement of $\mathcal{P}_0$.

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

*Summary.* Strategy semantics has given us a weakest $\perp$-preserving congruence (i.e. $[\mathcal{P}]_s^+$) for timed specification theories based on operators for (parallel) composition, conjunction, disjunction and quotient. Strategy semantics captures nicely the game-theoretical nature as well as the operational intuition of the specification theory. In the next section, we give a more declarative characterisation of the equivalence by means of timed traces.

## 4 Declarative Specification Theory

In this section, we develop a compositional specification theory based on timed traces. We introduce the concept of a timed-trace structure, which is an abstract representation for a timed component. The timed-trace structure contains essential information about the component, for checking whether it can be substituted with another in a safety and liveness preserving manner.

Given any TIOTS $\mathcal{P} = \langle I, O, S, s^0, \rightarrow \rangle$, we can extract three sets of traces from $(\mathcal{P}^{\perp})^{\top}$: $TP$ a set of timed traces leading to plain states; $TE$ a set of timed traces leading to the error state $\perp$; and $TM$ a set of timed traces leading to the magic state $\top$. $TE$ and $TM$ are extension-closed as $\top$ and $\perp$ are chaotic, while $TP$ is prefix-closed. Due to $\top/\perp$-completion, it is easy to verify $TE \cup TP \cup TM$ gives rise to the full set of timed traces $tA^*$; thus $TP$ and $TE$ are sufficient.

However, $TP$ and $TE$ contain more information than necessary for our substitutive refinement, which is designed to preserve $\perp$-freeness. For instance, adding any trace $tt \in TE$ to $TP$ should not change the semantics of the component. Based on a slight abstraction of the two sets, we can thus define a *trace structure* $\mathcal{TT}(\mathcal{P})$ as the semantics of $\mathcal{P}$.

**Definition 6 (Trace structure).** *$\mathcal{TT}(\mathcal{P}) := (I, O, TR, TE)$, where $TR := TE \cup TP$ the set of realisable traces. Obviously, $TR$ is prefix-closed.*

From hereon let $\mathcal{P}_0$ and $\mathcal{P}_1$ be two TIOTSs with trace structures $\mathcal{TT}(\mathcal{P}_i) := (I_i, O_i, TR_i, TE_i)$ for $i \in \{0, 1\}$. Define $\bar{i} = 1 - i$.

The substitutive refinement relation $\sqsubseteq$ in Section 2.3 can equally be characterised by means of trace containment. Consequently, $\mathcal{TT}(\mathcal{P}_0)$ can be regarded as providing an alternative encoding of the set $[\mathcal{P}_0]_s^+$ of strategies.

**Theorem 3.** *$\mathcal{P}_0 \sqsubseteq \mathcal{P}_1$ iff $TR_1 \subseteq TR_0$ and $TE_1 \subseteq TE_0$.*

We are now ready to define the timed-trace semantics for the operators of our specification theory. Intuitively, the timed-trace semantics mimic the synchronised product of the operational definitions in Section 2.3.

*Parallel composition.* The idea behind parallel composition is that the projection of any trace in the composition onto the alphabet of one of the components should be a trace of that component.

**Proposition 3.** *If $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\|$-composable, then $\mathcal{TT}(\mathcal{P}_0 \parallel \mathcal{P}_1) = (I, O, TR, TE)$ where $I = (I_0 \cup I_1) \setminus O$, $O = O_0 \cup O_1$ and the trace sets are given by:*

- $TE = \{tt \mid tt \upharpoonright tA_i \in TE_i \wedge tt \upharpoonright tA_{\bar{i}} \in TR_{\bar{i}}\} \cdot tA^*$
- $TR = TE \uplus \{tt \mid tt \upharpoonright tA_i \in (TR_i \setminus TE_i) \wedge tt \upharpoonright tA_{\bar{i}} \in (TR_{\bar{i}} \setminus TE_{\bar{i}})\}$

The above says $tt$ is an error trace if the projection of $tt$ on one component is an error trace, while the projection of $tt$ on the other component is a realisable trace. $tt$ is a realisable trace if $tt$ is either an error trace or a (strictly) plain trace. $tt$ is a (strictly) plain trace if the projections of $tt$ on to $\mathcal{P}_0$ and $\mathcal{P}_1$ are (strictly) plain traces.

*Disjunction.* From any composite state in the disjunction of two components, the composition should only be willing to accept inputs that are accepted by both components, but should accept the union of outputs. After witnessing an output enabled by only one of the components, the disjunction should behave like that component. Because of the way that $\bot$ and $\top$ work in Table 1, this loosely corresponds to taking the union of the traces from the respective components.

**Proposition 4.** *If $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\vee$-composable, then $\mathcal{TT}(\mathcal{P}_0 \vee \mathcal{P}_1) = (I, O, TR_0 \cup TR_1, TE_0 \cup TE_1)$, where $I = I_0 = I_1$ and $O = O_0 = O_1$.*

*Conjunction.* Similarly to disjunction, from any composite state in the conjunction of two components, the composition should only be willing to accept outputs that are accepted by both components, and should accept the union of inputs, until a stage when one of the component's input assumptions has been violated, after which it should behave like the other component. Because of the way that both $\bot$ and $\top$ work in Table 1, this essentially corresponds to taking the intersection of the traces from the respective components.

**Proposition 5.** *If $\mathcal{P}_0$ and $\mathcal{P}_1$ are $\wedge$-composable, then $\mathcal{TT}(\mathcal{P}_0 \wedge \mathcal{P}_1) = (I, O, TR_0 \cap TR_1, TE_0 \cap TE_1)$, where $I = I_0 = I_1$ and $O = O_0 = O_1$.*

*Quotient.* Quotient ensures its composition with the second component is a refinement of the first. Given the synchronised running of $\mathcal{P}_0$ and $\mathcal{P}_1$, if $\mathcal{P}_0$ is in a more refined state than $\mathcal{P}_1$, the quotient will try to rescue the refinement by taking $\top$ as its state (so that its composition with $\mathcal{P}_1$'s state will refine $\mathcal{P}_0$'s). If $\mathcal{P}_0$ is in a less or equally refined state than $\mathcal{P}_1$, the quotient will take the worst possible state without breaking the refinement.

**Proposition 6.** *If $\mathcal{P}_0$ dominates $\mathcal{P}_1$, then $\mathcal{TT}(\mathcal{P}_0 \% \mathcal{P}_1) = (I, O, TR, TE)$, where $I = I_0 \cup O_1$, $O = O_0 \setminus O_1$, and the trace sets satisfy:*

- $TE = TE_0 \cup \{tt \mid tt \upharpoonright tA_1 \notin TR_1\} \cdot tA^*$
- $TR = TE \uplus \{tt \mid tt \in (TR_0 \setminus TE_0) \wedge tt \upharpoonright tA_1 \in (TR_1 \setminus TE_1)\}$.

The above says $tt$ is an error trace if either $tt$ is an error trace in $\mathcal{P}_0$ or the projection of $tt$ on $\mathcal{P}_1$ is not a realisable trace. A strictly plain trace must have strictly plain projections onto $\mathcal{P}_0$ and $\mathcal{P}_1$.

Mirroring of trace structures is equally straightforward: $\mathcal{TT}(\mathcal{P}_0)^\neg = (O_0, I_0, tA^* \setminus TE_0, tA^* \setminus TR_0)$. Consequently, quotient can also be defined as the derived operator $(\mathcal{TT}(\mathcal{P}_0)^\neg \parallel \mathcal{TT}(\mathcal{P}_1))^\neg$.

## 5 Comparison with Related Works

Our framework can be seen as a linear-time alternative to the timed specification theories of [2] and [3], albeit with significant differences. The specification theory in [3] also introduces parallel, conjunction and quotient, but uses timed alternating simulation as refinement, which does not admit the weakest precongruence.

An advantage of [3] is the algorithmic efficiency of branching-time simulation checking and the implementation reported in [10].

The work of [2] on timed games also bears conceptual similarities, although they do not define conjunction and quotient. We adopt most of the game rules in [2], except that, due to our requirement that proposed delay moves are maximal delays allowed by a strategy, a play cannot have consecutive delay moves. This enables us to avoid the complexity of time-blocking strategies and blame assignment, but does not ensure non-Zenoness[8]. Secondly, we do not use timestop/semi-timestop to model time errors (i.e. bounded-liveness errors). Rather, we introduce the explicit inconsistent state $\perp$ to model both time and immediate (i.e. safety) errors. This enables us to avoid the complexity of having two transition relations and well-formedness of timed interfaces.

Based on linear time, our timed theory owes much to the pioneering work of trace theories in asynchronous circuit verification, such as Dill's trace theory [9]. Our mirror operator is essentially a timed extension of the mirror operator from asynchronous circuit verification [15]. The definition of quotient based on mirroring (for the untimed case) was first presented by Verhoeff as his Factorisation Theorem [14].

In comparison with our untimed theory [1], our timed extension requires new techniques (e.g. those related to timestop) to handle delay transitions since time can be modelled neither as input nor as output. In the timed theory, the set of realisable traces ($TR$) is not required to be input-enabled, which is necessary for the set of untimed traces in [1]. Thus, the domain of trace structures is significantly enlarged. Furthermore, the timed theory supports the modelling of liveness assumptions/guarantees, with the checking of such violations reducing to $\perp$-reachability. Therefore, finite traces suffice to model and verify liveness properties, whereas in contrast, the untimed theory must employ infinite traces to treat liveness in a proper way.

We briefly mention other related works, which include timed modal transition systems [11,12], the timed I/O model [5,13] and embedded systems [18,19].

## 6 Conclusions

We have formulated a rich compositional specification theory for components with real-time constraints, based on a linear-time notion of substitutive refinement. The operators of hiding and renaming can also be defined, based on our previous work [8]. We believe that our theory can be reformulated as a timed extension of Dill's trace theory [9]. Future work will include an investigation of realisability and assume-guarantee reasoning.

**Acknowledgments.** The authors are supported by EU FP7 project CONNECT, ERC Advanced Grant VERIWARE and EPSRC project EP/F001096.

[8] Zeno behaviours (infinite action moves within finite time) in a play are not regarded as abnormal behaviours in our semantics.

## References


1. Chen, T., Chilton, C., Jonsson, B., Kwiatkowska, M.: A Compositional Specification Theory for Component Behaviours. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 148–168. Springer, Heidelberg (2012)
2. de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed Interfaces. In: Sangiovanni-Vincentelli, A.L., Sifakis, J. (eds.) EMSOFT 2002. LNCS, vol. 2491, pp. 108–122. Springer, Heidelberg (2002)
3. 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, pp. 91–100. ACM (2010)
4. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes 26, 109–120 (2001)
5. Kaynar, D.K., Lynch, N.A., Segala, R., Vaandrager, F.W.: Timed I/O Automata: A mathematical framework for modeling and analyzing real-time systems. In: RTSS (2003)
6. Lim, W.: Design methodology for stoppable clock systems. Computers and Digital Techniques, IEE Proceedings E 133, 65–72 (1986)
7. Moore, S.W., Taylor, G.S., Cunningham, P.A., Mullins, R.D., Robinson, P.: Using stoppable clocks to safely interface asynchronous and synchronous subsystems. In: AINT (Asynchronous INTerfaces) Workshop, Delft, Netherlands (2000)
8. Wang, X., Kwiatkowska, M.Z.: On process-algebraic verification of asynchronous circuits. Fundam. Inform. 80, 283–310 (2007)
9. Dill, D.L.: Trace theory for automatic hierarchical verification of speed-independent circuits. ACM distinguished dissertations. MIT Press (1989)
10. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: ECDAR: An Environment for Compositional Design and Analysis of Real Time Systems. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 365–370. Springer, Heidelberg (2010)
11. Bertrand, N., Legay, A., Pinchinat, S., Raclet, J.-B.: A Compositional Approach on Modal Specifications for Timed Systems. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 679–697. Springer, Heidelberg (2009)
12. Cerans, K., Godskesen, J.C., Larsen, K.G.: Timed Modal Specification - Theory and Tools. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 253–267. Springer, Heidelberg (1993)
13. Berendsen, J., Vaandrager, F.W.: Compositional Abstraction in Real-Time Model Checking. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 233–249. Springer, Heidelberg (2008)
14. Verhoeff, T.: A Theory of Delay-Insensitive Systems. PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology (1994)
15. Zhou, B., Yoneda, T., Myers, C.: Framework of timed trace theoretic verification revisited. IEICE Trans. on Information and Systems 85, 1595–1604 (2002)
16. Chilton, C., Kwiatkowska, M., Wang, X.: Revisiting timed specification theories: A linear-time perspective. Technical Report RR-12-04, Department of Computer Science, University of Oxford (2012)
17. Wang, X.: Maximal Confluent Processes. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 188–207. Springer, Heidelberg (2012)
18. Thiele, L., Wandeler, E., Stoimenov, N.: Real-time interfaces for composing real-time systems. In: EMSOFT (2006)
19. Lee, I., Leung, J., Song, S.: Handbook of Real-Time and Embedded Systems. Chapman (2007)