Timed Interfaces*

Luca de Alfaro¹, Thomas A. Henzinger², and Mariëlle Stoelinga¹

¹ Computer Engineering, University of California, Santa Cruz
{luca,marielle}@so.eucsc.edu
² EECS, University of California, Berkeley
tah@eecs.berkeley.edu

Abstract. We present a theory of timed interfaces, which is capable of specifying both the timing of the inputs a component expects from the environment, and the timing of the outputs it can produce. Two timed interfaces are compatible if there is a way to use them together such that their timing expectations are met. Our theory provides algorithms for checking the compatibility between two interfaces and for deriving the composite interface; the theory can thus be viewed as a type system for real-time interaction. Technically, a timed interface is encoded as a timed game between two players, representing the inputs and outputs of the component. The algorithms for compatibility checking and interface composition are thus derived from algorithms for solving timed games.

1 Introduction

A formal notion of component interfaces provides a way to describe the interaction between components, and to verify the compatibility between components automatically. Traditional type systems capture only the data dimension of interfaces (“what are the value constraints on data communicated between components?”). We have developed an approach, called interface theories [dAH01a,dAH01b], which can be viewed as a behavioral type system that also captures the protocol dimension of interfaces (“what are the temporal ordering constraints on communication events between components?”). This paper extends this formalism to capture, in addition, the timing dimension of interfaces (“what are the real-time constraints on communication events between components?”). This permits, for example, the specification and compatibility check of component interactions based on time-outs. Timed interfaces support the component-based design of real-time systems in the following ways:

Component interface specification. A component is an open system that expects inputs and provides outputs to the environment. An interface specifies how a component interacts with its environment, by describing both the assumptions made by the component on the inputs and the guarantees provided by the

* This research was supported in part by the NSF CAREER award CCR-0132780, the NSF grant CCR-9988172 the AFOSR MURI grant F49620-00-1-0327, the DARPA PCES grant F33615-00-C-1693, the MARCO GSRC grant 98-DT-660, and the ONR grant N00014-02-1-0671.
component on the outputs. Timed interfaces can refer to the timing of the input and output events. An interface is well-formed as long as there is some environment that satisfies the input assumptions made by the component; otherwise, the component would not be usable in any design.

**Interface compatibility checking.** When two components are composed, we can check that the components satisfy each other’s assumptions. Since the result of composition is generally still an open system, it may depend on the environment whether or not this is the case. Two interfaces are compatible if their composition is well-formed, i.e., if there exists an environment that makes them work together. The composition yields a new interface for the composite system that specifies the derived input assumptions required to make the original components working together, as well as the resulting output guarantees.

An (untimed or timed) interface is naturally modeled as a game between two players, Output and Input. Player Output represents the component; the moves of Output represent the possible outputs that the component may generate (the output guarantees); player Input represents the environment; the moves of Input represent the inputs that the system accepts from the environment (the input assumptions). For instance, a functional type $D \rightarrow D'$ in a programming language is an interface that corresponds to a one-shot game. The Input player provides inputs that can be accepted (values in $D$) and the Output player provides outputs that can be generated (values in $D'$). If the sets of legal inputs and possible outputs can change dynamically over time, then the interface is naturally modeled as an iterative game played on a set of states [Abr96], where the moves —i.e., the acceptable inputs and possible outputs— may depend on the state of the system. Such an interface is well-formed if the Input player has a winning strategy in the game, i.e., the environment can meet all input assumptions. For timed interfaces, we need the additional well-formedness condition that a player must not achieve its goal by blocking time forever.

The game-theoretic view of interfaces becomes most apparent in their composition. When two interfaces are composed, the combined interface may contain error states. These occur when one component interface can generate an output that is not a legal input for the other component interface, showing that the first component violates an input assumption of the second. In addition, our timed games give rise to time errors, where one of the players cannot let time pass. Two interfaces are compatible if there is a way for the Input player, who chooses the inputs of the composite interface, to avoid all errors. If so, then there exists an environment of the combined system which makes both components satisfy each other’s input assumptions.

Different theories of timed interfaces arise depending on the details of how timed games are defined. For example, communication can be through actions or shared variables, and composition can be synchronous or asynchronous. The main contribution of this paper does not lie in such details of the formalism (which may be changed), but in the notion of interface of a real-time component as a timed game. This notion sets our interfaces apart from the type systems for protocols of [RR01,CRR02], from message-sequence charts [RGG96], and from
traditional models for timed systems such as timed automata [AD94, MMT91]. Many models are unable to express input assumptions and postulate that a component must work in all environments; this is the input-enabled approach of, e.g., [AH97, SCSA98]. Models that can encode input assumptions, such as process algebras, often phrase the compatibility question as a graph (rather than game) question, in which input and output play the same role: two components are considered compatible if they cannot reach a deadlock [RR01]. In our game-based approach, input and output play dual roles: two components are compatible if there is some input behavior such that, for all output behaviors, no incompatibility arises. This notion captures the idea that an interface can be useful as long as it can be used in some design. In this, interfaces are close to types in programming languages, to trace theory [Dil88], and to the semantics of interaction [Abr96].

2 Timed Interfaces as Timed Games: Preview

In a timed interface, the Input and Output players have two kinds of moves: immediate moves, which represent events sent or received by the interface, and timed moves, that consist in an amount of time that the players propose to spend idle. We assume a time domain $\mathbb{T}$; suitable choices for $\mathbb{T}$ are the nonnegative reals $\mathbb{R}_{\geq 0}$, or the nonnegative integers $\mathbb{N}$. The successor state is determined as follows. When Input chooses $t_I \in \mathbb{T}$ and Output chooses $t_O \in \mathbb{T}$, the global time will advance by $\min\{t_I, t_O\}$; if a player chooses an immediate move and the other a timed move, the immediate move prevails; if both players choose immediate moves, one of them occurs nondeterministically [MPS95, AMPS98].

Only game outcomes along which global time diverges are physically meaningful. Obviously, each player is capable of blocking the progress of time by playing a sequence of timed moves whose summation converges (so-called Zeno behavior). To rule out such behavior, we require a well-formedness criterion, which states that at all reachable states, each player can ensure that time progresses unless the other player blocks its progress. The composition of timed interfaces may give rise to two kinds of error states: immediate error states, where one interface emits an output that cannot be accepted by the other, and time error states, where the well-formedness criterion is violated. Two interfaces are compatible if there is a strategy for Input in the combined interface that avoids all errors. The composite interface is obtained by restricting the input moves (i.e., the accepted environments) such that the error states are not entered.

We illustrate these concepts through a simple example from scheduling. The interfaces are modeled as timed interface automata. This is a syntax derived from timed automata [AD94], however, the syntax is interpreted as a (timed) game, rather than as a (timed) transition system. In particular, timed automata use invariants for specifying an upper bound for the advancement of time at a location. Timed interface automata have two kinds of invariants: input invariants, which specify upper bounds for the timed moves played by Input, and output invariants, which specify upper bounds for the timed moves played by Output.
Fig. 1: Timed interface automata for periodic job scheduling.

**Example: scheduling a periodic job.** The timed interface automaton *Caller* shown in Figure 1(a) represents an application that activates a periodic job every 10 seconds. Each time the job is activated, it must terminate between 6 and 8 seconds. The clock $x$ measures the time elapsed since the last activation of the job. At the initial location $p_0$, the Output player can play any timed move $\Delta$ such that $\Delta + x \leq 10$, that is, it can advance time only up to the point where $x = 10$. Once $x = 10$, in order not to block progress of time, Output must eventually play the move $js$ (job start); i.e., the output invariant $x \leq 10$ expresses a deadline for the occurrence of the $js$ action. Since the input invariant associated with $p_0$ is always true, the Input player can play any $\Delta \in \Gamma$ (to reduce clutter, invariants that are always true are omitted from the figures). The move $js$ resets $x$ to 0, ensuring that $x$ counts the time from the start of a periodic job activation.

At location $p_1$, Output can play arbitrary timed moves, since the output invariant is true. The input invariant $x \leq 8$ prevents the Input player from advancing time beyond $x = 8$. So, to let time progress, Input must play the move $jf$ (job finished) somewhere between $6 \leq x \leq 8$. In particular, $x \leq 8$ is a deadline for the reception of action $jf$.

**Composition.** The timed interface automaton *Caller* can be composed with the timed interface automaton *Job*, which models the interface of the job to be
executed. The job starts when the input move $js$ (job start) is received; the job then waits for the input move $ms$ (machine start) from a scheduler. Once $ms$ is received, the job executes for 4 seconds (clock $y$ keeps track of the execution time), after which it first indicates to the scheduler that the machine is no longer used (output move $mf$, machine finished), and then finishes (output move $jf$, job finished). Note how, at state $q_2$, the input invariant is true, indicating that Input can play any timed move, while the output invariant is $y \leq 4$, indicating that Output can let time progress only until $y = 4$. Again, to avoid blocking the progress of time, Output must eventually play the move $mf$. Similarly, Output is forced to play $jf$ at state $q_3$.

To compute the composition $\text{Caller} \parallel \text{Job}$, we first compute their product $\text{Caller} \otimes \text{Job}$, shown in Figure 1(c), which represents the joint behavior arising from $\text{Caller}$ and $\text{Job}$. Moves synchronize in multi-way, broadcast fashion: the synchronization between corresponding input and output moves gives rise to an output move. To compute the composition $\text{Caller}$ from $\text{Caller} \otimes \text{Job}$, note that the product contains error states: at location $(p_1, q_3)$, if $\text{err}_3 : \neg(6 \leq x \leq 8)$ holds, then interface $\text{Job}$ can generate the output $jf$, which cannot be accepted by $\text{Caller}$. However, the Input player can avoid the error states by choosing the timing of the input $ms$. To see this, note that at location $(p_1, q_2)$ the set of uncontrolled states from which Input cannot avoid reaching the error states is described by the predicate $\text{uncontr}_2 : \exists \Delta \in T.(y + \Delta = 4 \land \neg(6 \leq x + \Delta \leq 8))$, or after simplification, $\text{uncontr}_2 : \neg(2 \leq x - y \leq 4)$. Hence, we conjoin $2 \leq x - y \leq 4$ to the input invariant of $(p_1, q_1)$, which yields $2 \leq x - y \leq 4 \land x \leq 8$ as the input invariant in $\text{Caller} \parallel \text{Job}$ (see Figure 1(d)). We simplify this to $2 \leq x - y \leq 4$, because the states where $x > 8$ cannot be reached: from $y \leq 4$ (the output invariant) and $x - y \leq 4$ (a portion of the input invariant) follows $x \leq 8$.

Now, to avoid entering uncontrolled states at location $(p_1, q_2)$ we have to restrict the time during which the input move $ms$ can be received: the most liberal restriction consists in requiring $2 \leq x \leq 4$, which is added to enabling condition of $ms$. Finally, consider the location $(p_1, q_1)$, whose input invariant is $x \leq 8$. To ensure progress of time beyond $x = 8$, Input must eventually play the move $ms$. However, this move is available only when $2 \leq x \leq 4$. Thus, if $x > 4$, the Input player does not have a strategy that ensures that time diverges unless blocked by the Output player. Therefore, $x > 4$ indicates time error states, which are ruled out by strengthening the input invariant to $x \leq 4$. Notice how the composition of timed interfaces effects the composition of timing requirements: the requirement of $\text{Caller}$ that the job must be completed between 6 and 8 seconds gives rise to the requirement for the scheduler that the machine is started between 2 and 4 seconds after the job start.

3 Timed Interfaces as Timed Games: Definitions

We model interfaces as timed games between two players, Input and Output, abbreviated $I$ and $O$. 
Definition 1 (timed interfaces) A timed interface is a tuple \( \mathcal{P} = (\mathcal{S}_P, s^\text{init}_P, \text{Acts}^I_P, \text{Acts}^O_P, \rho^I_P, \rho^O_P) \) consisting of the following components.

- \( \mathcal{S}_P \) is a set of states.
- \( s^\text{init}_P \in \mathcal{S}_P \) is the initial state.
- \( \text{Acts}^I_P \) and \( \text{Acts}^O_P \) are sets of immediate input and output actions, respectively. These sets must be disjoint from each other, and disjoint from the time domain \( \mathcal{T} \). We denote by \( \text{Acts}_P = \text{Acts}^I_P \cup \text{Acts}^O_P \) the set of all immediate actions, by \( \Gamma^I_P = \text{Acts}^I_P \cup \mathcal{T} \) the set of all input actions, and by \( \Gamma^O_P = \text{Acts}^O_P \cup \mathcal{T} \) the set of all output actions. The elements in \( \mathcal{T} \) are timed actions.
- \( \rho^I_P \subseteq \mathcal{S}_P \times \Gamma^I_P \times \mathcal{S}_P \) is the input transition relation, and \( \rho^O_P \subseteq \mathcal{S}_P \times \Gamma^O_P \times \mathcal{S}_P \) is the output transition relation. We often write an element \((s, \alpha, s')\) of a transition relation as \( s \xrightarrow{\alpha} s' \), and call it a step. Given a state \( s \in \mathcal{S}_P \) and a player \( \gamma \in \{I, O\} \), the set of moves of player \( \gamma \) at \( s \) is \( \Gamma^\gamma_P(s) = \{ \alpha \in \Gamma^\gamma_P \mid \exists s' \in \mathcal{S}_P . (s \xrightarrow{\alpha} s') \in \rho^\gamma_P \} \).

We require the transition relations to be deterministic: for \( \gamma \in \{I, O\} \), if \((s, a, s') \in \rho^\gamma_P \) and \((s, a, s'') \in \rho^\gamma_P \), then \( s' = s'' \). Furthermore, we require time determinism for time steps over both relations: for all \( \Delta \in \mathcal{T} \), if \((s, \Delta, s') \in \rho^I_P \) and \((s, \Delta, s'') \in \rho^I_P \), then \( s' = s'' \). Time steps of duration 0 do not leave the state: for \( \gamma \in \{I, O\} \), if \((s, 0, s') \in \rho^\gamma_P \), then \( s = s' \). We also require Wang’s Axiom [Y90]: for all \( \Delta, \Delta' \in \mathcal{T} \) with \( \Delta' \leq \Delta \), we have \((s, \Delta, s'') \in \rho^I_P \) iff there is a state \( s' \) such that both \((s, \Delta', s') \in \rho^I_P \) and \((s', \Delta - \Delta', s'') \in \rho^I_P \). Finally, if there are any immediate actions available to a player in a state, then also the timed action 0 is available: for \( \gamma \in \{I, O\} \), if \( \Gamma^\gamma_P(s) \neq \emptyset \), then \((s, 0, s) \in \rho^\gamma_P \). 

The game proceeds as follows. At a state \( s \in \mathcal{S}_P \), Input chooses a move from \( \Gamma^I_P(s) \), and Output chooses a move from \( \Gamma^O_P(s) \). If no moves are available for a player, that player will automatically lose the game. If two moves are played, then these determine both the successor state and the player \( bl \) that is blamed for having played first; assigning the blame is important in establishing whether a player is blocking the progress of time [AH97]. The following definition is asymmetric: when Input and Output play the same timed move, the Output player is blamed. As we will illustrate later, this asymmetry is necessary to capture the cause-effect relationship between outputs and inputs.

Definition 2 (move outcomes) For all states \( s \in \mathcal{S}_P \) and moves \( \alpha_I \in \Gamma^I_P(s) \) and \( \alpha_O \in \Gamma^O_P(s) \), the outcome \( \delta_P(s, \alpha_I, \alpha_O) \) of \( \alpha_I \) and \( \alpha_O \) at \( s \) is the set of triples \((\alpha, \rho^I_P, bl)\) such that \((s \xrightarrow{\alpha} s') \in \rho^I_P \cup \rho^O_P \) and \( \alpha \in \Gamma^I_P \cup \Gamma^O_P \) and \( bl \in \{I, O\} \) are obtained as follows.

- If \( \alpha_I, \alpha_O \in \mathcal{T} \), then \( \alpha = \min\{\alpha_I, \alpha_O\} \). Moreover, \( bl = I \) if \( \alpha_I \prec \alpha_O \), and \( bl = O \) otherwise.
- If \( \alpha_I \in \text{Acts}_P \) and \( \alpha_O \in \mathcal{T} \), then \( \alpha = \alpha_I \) and \( bl = I \).
- If \( \alpha_O \in \text{Acts}_P \) and \( \alpha_I \in \mathcal{T} \), then \( \alpha = \alpha_O \) and \( bl = O \).
- If \( \alpha_I, \alpha_O \in \text{Acts}_P \), then either \( \alpha = \alpha_I \) and \( bl = I \), or \( \alpha = \alpha_O \) and \( bl = O \).
As usual, the players choose their moves according to strategies that may depend on the history of the game. Our strategies are partial functions, rather than total ones, because the sets of moves available to the players at a state can be empty. Furthermore, if both players choose immediate actions, then the outcome is nondeterministic. Consequently, the possible outcomes of two strategies form a set of finite and infinite sequences. A state is reachable in the game if it can be reached by some outcome of some input and output strategies.

**Definition 3 (strategy outcomes)** A strategy for player \( \gamma \in \{ I, O \} \) is a partial function \( \pi^\gamma : S^\gamma \rightarrow I^\gamma_P \) that associates, with every finite sequence of states \( \overline{s} \in S^\gamma \), whose final state is \( s \), a move \( \pi^\gamma(\overline{s}) \in I^\gamma_P(s) \) provided that \( I^\gamma_P(s) \neq \emptyset \); otherwise \( \pi^\gamma(\overline{s}) \) is undefined. Let \( I^\gamma_P \) be the set of strategies for player \( I \), and let \( O^\gamma_P \) be the set of strategies for player \( O \). Given a state \( s \in S^\gamma \), an input strategy \( \pi^I \in I^\gamma_P \), and an output strategy \( \pi^O \in O^\gamma_P \), the set of outcomes \( \hat{\delta}_P(s, \pi^I, \pi^O) \) of \( \pi^I \) and \( \pi^O \) from \( s \) consists of all finite and infinite sequences

\[ \sigma = (s_0, b_{l_0}, a_1, (s_1, b_1), a_2, (s_2, b_2), \ldots) \]  

such that (1) \( s_0 = s \); (2) \( b_{l_0} \in \{ I, O \} \); (3) if \( |\sigma| < \infty \), then \( \sigma \) ends in a pair \( (a_k, b_{k}) \) such that \( I^\gamma_P(a_k) = \emptyset \) or \( O^\gamma_P(b_k) = \emptyset \); and (4) for all \( n < |\sigma| \), we have \( (s_{n+1}, a_{n}, (s_{n+1}, b_{n+1})) \in \delta_P(s_n, \pi^I(s_{n+1}), \pi^O(s_{n+1})) \), where \( \sigma_{0:n} \) denotes the prefix \( (s_0, b_{l_0}), a_1, \ldots, (s_n, b_n) \) of \( \sigma \) with length \( n \).

A state \( s \in S^\gamma \) is reachable in \( P \) if there are two strategies \( \pi^I \in I^\gamma_P \) and \( \pi^O \in O^\gamma_P \), and \( k \geq 0 \), such that \( s = s_k \) for some outcome \( (s_0, b_{l_0}), a_1, (s_1, b_1), a_2, (s_2, b_2), \ldots \in \hat{\delta}_P(s^\gamma_{0:n}, \pi^I, \pi^O) \).

### 3.1 Well-formedness of timed interfaces

A timed interface is well-formed if from every reachable state (i) Input has a strategy such that either time diverges, or Output is always to blame beyond some point; and (ii) symmetrically, Output has a strategy such that either time diverges, or Input is always to blame beyond some point. To give the precise definitions, let \( time(\alpha) = \alpha \) for \( \alpha \in T \), and \( time(\alpha) = 0 \) otherwise.

**Definition 4 (time divergence and time blocking)** Let \( \sigma = (s_0, b_{l_0}), a_1, (s_1, b_1), a_2, (s_2, b_2), \ldots \) be an outcome of a game in \( P \). We define \( \sigma \models t_{\text{div}} \) if the accumulated time in \( \sigma \) is infinite, that is, \( \sum_{k=1}^{\sigma |} time(a_k) = \infty \). For \( \gamma \in \{ I, O \} \), we define \( \sigma \models \text{blame}^\gamma \) if either \( \sigma \) is finite and \( I^\gamma_P(s) = \emptyset \) for \( \sigma \)'s last state \( s \), or \( \sigma \) is infinite and there is \( k \geq 0 \) such that \( b_{i} = \gamma \) for all \( i > k \). For a set \( U \subseteq S^\gamma \) of states, we define \( \sigma \models \Box U \) if \( s_k \in U \) for all \( k \geq 0 \).

A state of a timed interface \( P \) is live in a set \( U \) of states if both players have strategies to stay forever in \( U \) and let time advance, unless the other player can be blamed for blocking the progress of time. Again, the game is not played symmetrically: Input can choose its strategy after Output, which shows that the game is turn-based (first Output chooses its move, then Input does).

**Definition 5 (live states and well-formedness)** Let \( U \subseteq S^\gamma \) be a set of states. A state \( s \in S^\gamma \) is \( I \)-live in \( U \) if Input can win the game with goal \( (t_{\text{div}} \lor \)
\( \text{blame}^O \) \( \land \Box U \); that is, if for all strategies \( \pi^O \in \Pi^O_p \) there is a strategy \( \pi^I \in \Pi^I_p \) such that \( \sigma \models (t_{\div} \lor \text{blame}^O) \land \Box U \) for all outcomes \( \sigma \in \delta_p(s, \pi^I, \pi^O) \). A state \( s \in S_p \) is live in \( U \) if Output can win the game with goal \( (t_{\div} \lor \text{blame}^I) \land \Box U \); that is, if there is a strategy \( \pi^O \in \Pi^O_p \) such that for all strategies \( \pi^I \in \Pi^I_p \) and outcomes \( \sigma \in \delta_p(s, \pi^I, \pi^O) \), we have \( \sigma \models (t_{\div} \lor \text{blame}^I) \land \Box U \). A state \( s \in S_p \) is live in \( U \) if it is both I-live and O-live in \( U \).

The timed interface \( P \) is well-formed in \( U \) if all reachable states of \( P \) are live in \( U \). The timed interface \( P \) is well-formed if it is well-formed in \( S_p \).

In particular, for all reachable states \( s \) of a well-formed timed interface \( P \), both \( \Gamma^I_p(s) \neq \emptyset \) and \( \Gamma^O_p(s) \neq \emptyset \). Only well-formed timed interfaces represent valid interface specifications. For this reason, we will only define the composition of well-formed timed interfaces.

### 3.2 Product and composition of timed interfaces

Two timed interfaces \( P \) and \( Q \) are composable if \( \text{Acts}^O_P \cap \text{Acts}^O_Q = \emptyset \). The shared actions of \( P \) and \( Q \) are given by \( \text{shared}(P, Q) = \text{Acts}^O_P \cap \text{Acts}^O_Q \). For two composable timed interfaces \( P \) and \( Q \), their composition \( P \| Q \) is computed in two steps: first, we form the product \( P \otimes Q \) together with the set \( i\text{-errors}(P, Q) \) of immediate error states; then, \( P \| Q \) is obtained by strengthening the input invariants of \( P \otimes Q \) to make it well-formed in \( S_{P \otimes Q} \setminus i\text{-errors}(P, Q) \). The product \( P \otimes Q \) represents the joint behavior of \( P \) and \( Q \), in which \( P \) and \( Q \) synchronize on the input timed moves, on the output timed moves, and on the shared actions, and behave independently otherwise.

**Definition 6 (product)** Given two composable timed interfaces \( P_1 \) and \( P_2 \), the product \( P_1 \otimes P_2 \) is the timed interface that consists of the following components.

- \( S_{P_1 \otimes P_2} = S_{P_1} \times S_{P_2} \), and \( s^\text{init}_{P_1 \otimes P_2} = (s^\text{init}_{P_1}, s^\text{init}_{P_2}) \).
- \( \text{Acts}^I_{P_1 \otimes P_2} = \text{Acts}^I_{P_1} \cup \text{Acts}^I_{P_2} \setminus \text{shared}(P_1, P_2) \), and \( \text{Acts}^O_{P_1 \otimes P_2} = \text{Acts}^O_{P_1} \cup \text{Acts}^O_{P_2} \).
- \( \rho^I_{P_1 \otimes P_2} \) is the set of transitions \( (s_1, s_2) \xrightarrow{\alpha} (s'_1, s'_2) \) such that for \( i = 1, 2 \); if \( \alpha \in \Gamma^I_{P_i} \), then \( (s_i \xrightarrow{\alpha} s'_i) \in \rho^I_{P_i} \); otherwise \( (s_i \not\xrightarrow{\alpha} s'_i) \in \rho^I_{P_i} \).
- \( \rho^O_{P_1 \otimes P_2} \) is the set of transitions \( (s_1, s_2) \xrightarrow{\alpha} (s'_1, s'_2) \) such that for \( i = 1, 2 \); if \( \alpha \in \Gamma^O_{P_i} \), then \( (s_i \xrightarrow{\alpha} s'_i) \in \rho^O_{P_i} \); if \( \alpha \in \text{Acts}^I_{P_i} \), then \( (s_i \xrightarrow{\alpha} s'_i) \in \rho^I_{P_i} \); and otherwise \( (s_i \not\xrightarrow{\alpha} s'_i) \in \rho^O_{P_i} \).
Definition 7 (error states) Let $\mathcal{P}$ and $\mathcal{Q}$ be two well-formed and composable timed interfaces.

- **Immediate error states.** We say that a state $(s, t) \in S_{\mathcal{P} \otimes \mathcal{Q}}$ is an immediate error state if there is an action $a \in \text{shared}(\mathcal{P}, \mathcal{Q})$ such that $(s \xrightarrow{a} s') \in \rho^\mathcal{P}$ for some state $s'$, but $(t \xrightarrow{a} t') \notin \rho^\mathcal{Q}$ for all states $t', \text{ or such that } (t \xrightarrow{a} t') \in \rho^\mathcal{Q}$ for some $t'$, but $(s \xrightarrow{a} s') \notin \rho^\mathcal{P}$ for all $s'$. We denote by $i$-errors$(\mathcal{P}, \mathcal{Q}) \subseteq S_{\mathcal{P} \otimes \mathcal{Q}}$ the set of immediate error states.

- **Time error states.** We say that a state $(s, t) \in S_{\mathcal{P} \otimes \mathcal{Q}}$ is a time error state if $(s, t)$ is reachable in $\mathcal{P} \otimes \mathcal{Q}$, but it is not $I$-live in $S_{\mathcal{P} \otimes \mathcal{Q}} \setminus i$-errors$(\mathcal{P}, \mathcal{Q})$. We denote by $t$-errors$(\mathcal{P}, \mathcal{Q}) \subseteq S_{\mathcal{P} \otimes \mathcal{Q}}$ the set of time error states.

Note that the reachable immediate error states are a subset of the time error states. The composition $\mathcal{P} \parallel \mathcal{Q}$ is obtained by restricting the input behavior of $\mathcal{P} \otimes \mathcal{Q}$ to avoid all time error states. We restrict the input behavior only, leaving the output behavior unchanged, because when composing interfaces we can strengthen their input assumptions to ensure that no incompatibility arises, but we cannot modify their output behavior.

Definition 8 (compatibility and composition) Let $\mathcal{P}$ and $\mathcal{Q}$ be two well-formed and composable timed interfaces. The interfaces $\mathcal{P}$ and $\mathcal{Q}$ are compatible if $(s^\mathcal{P}_{\text{init}}, s^\mathcal{Q}_{\text{init}}) \notin t$-errors$(\mathcal{P}, \mathcal{Q})$. If $\mathcal{P}$ and $\mathcal{Q}$ are compatible, then the composition $\mathcal{P} \parallel \mathcal{Q}$ is defined by restricting the input transition relation, so that no error states are entered. Formally, abbreviating $U = S_{\mathcal{P} \otimes \mathcal{Q}} \setminus t$-errors$(\mathcal{P}, \mathcal{Q})$, we define $\rho^\mathcal{P}_{\mathcal{P} \parallel \mathcal{Q}} = \rho^\mathcal{P}_{\mathcal{P} \otimes \mathcal{Q}} \cap (U \times \text{Acts}_{\mathcal{P} \otimes \mathcal{Q}} \times U)$; all other components of $\mathcal{P} \parallel \mathcal{Q}$ are defined as in $\mathcal{P} \otimes \mathcal{Q}$. □

3.3 Discussion

**Well-formedness.** The composition of timed interfaces that are not well-formed may yield undesirable results, and hence is not defined in our theory. To illustrate this point, consider the interfaces in Figure 2. The time steps are represented as follows: for player $\gamma \in \{I, O\}$, if state $s$ has label $\gamma : \Delta = 0$, then only the time step $(s, 0, s)$ is available for $\gamma$; if $s$ has no $\gamma$-label, then all time steps $(s, \Delta, s)$ with $\Delta \in T$ are available for $\gamma$ at $s$. In interface $\mathcal{P}_1$, there is no deadline associated with the immediate move $a$: Output can play it at any time, or not at all. Similarly, $\mathcal{P}_2$ does not associate a deadline to $a$: Input can play it at any
Fig. 3: Timed interfaces illustrating why two transition relations are needed.

time, or not at all. Note that $P_2$ is not well-formed. This is because Output can only play the timed move 0 at state $t_0$, so $t_0$ is not $O$-live: if Input also plays the timed move 0, Output can neither let time diverge, nor blame Input. Consider now the product $P_1 \otimes P_2$. The product specifies that the output move $a$ must be played at time 0, even though no such deadline for $a$ was present at $P_1$ or $P_2$. The problem, intuitively, is that the deadline of 0 for Output in $P_2$ does not apply to any output move. When an unrelated output move becomes present, such as $a$ in $P_1 \otimes P_2$, the deadline is improperly transferred to the move. The well-formedness condition requires that a player has a deadline only if it also has an action that can satisfy the deadline, thus preventing such “deadline transfers.”

Two transition relations. If we use a single transition relation, rather than one for Input and one for Output, then both players would share the same timed moves at a state. The following example shows that, if we do so, the composition of interfaces may again introduce timing requirements that are not present in the component interfaces. In particular, composition may not be associative. Consider the timed interfaces $P_3$ and $P_4$ in Figure 3, where the timing constraints apply to both Input and Output moves. The interfaces $P_3$ and $P_4$ are compatible. In state $s_0$, the interface $P_4$ has to take an input action at time 0. This input need not be provided by $P_3$, because $P_3$ is not forced to take its $a!$ action. Nevertheless, $P_3$ and $P_4$ work together in an environment that provides a $c$ action at time 0. Now consider the product $P_3 \otimes P_4$ in Figure 3(c); note that $P_3 \otimes P_4 = P_3 \parallel P_4$, because the product is well-formed. The composition $P_3 \parallel P_4$ specifies that, if $c$ is not received at 0, then output $a$ is produced at time 0. As a result, $P_3 \parallel P_4$ also works in environments that never provide a $c$ input. More formally, let $Q$ be the interface that has an output move $c$, which can never be taken. Then $(P_3 \parallel P_4)$ and $Q$ are compatible. However, $P_4 \parallel Q$ (Figure 3(e)) and $P_3$ are not compatible. Hence, the compatibility of $P_3, P_4$, and $Q$ depends on the order in which they are composed.

Game asymmetry. This example uses the timed interface automata notation as in Section 2, to be introduced formally in the next section. Consider the
Fig. 4: Timed interfaces illustrating why \( O \) is blamed and why \( I \) plays second.

automata \( A, A_1, \) and \( A_2 \) in Figure 4, where \( \prec \in \{<, \leq\} \). Note that \( A = A \otimes A_2 \). Since \( A_1 \) provides an output \( a \) within the deadline required by \( A_2 \), the automata \( A_1 \) and \( A_2 \) should be compatible, and \( A \) should be well-formed. Consider first the case when \( \prec \) is \( \leq \). If Input is blamed when it plays the same time move as Output, then \( A \) is not well-formed. In fact, Output can play the strategy of advancing time until \( x = 1 \), and thereafter always play timed move 0. Input cannot let time progress, nor can it blame Output. Hence, any state \( (s_0, x) \) with \( x \leq 1 \) is not \( I \)-live, and \( A \) is not well-formed. Consider now the case when \( \prec \) is \(<\). If Input cannot play second, then for each strategy \( \pi^f \) of Input, there is a strategy \( \pi^O \) for Output that plays later, i.e., a strategy \( \pi^O \) such that \( \pi^f(\pi) < \pi^O(\pi) < 1 \) for all histories \( \pi \). Then the outcome of this strategy neither is time divergent nor does it blame \( O \). If, on the other hand, \( I \) plays second, then it can win the game by playing later than \( O \).

4 Timed Interface Automata

Timed interfaces provide a finite representation for timed games and serve as a basis on which the algorithms for compatibility checking and composition operate. Their syntax recalls that of timed automata [AD94]. In particular, timed interface automata use clock variables in order to keep track of the amount of time elapsed. The value of these variables can be reset to 0 when immediate actions occur, and otherwise increase with unit rate. Let \( \mathcal{X} \) be a set of variables over the time domain \( \mathbb{T} \). A clock condition over \( \mathcal{X} \) is a boolean combination of formulas of the form \( x < c \) or \( x - y < c \), where \( c \) is an integer, \( x, y \in \mathcal{X} \), and \( \prec \) is either of \( < \) or \( \leq \). We denote the set of all clock conditions over \( \mathcal{X} \) by \( \Xi[\mathcal{X}] \).

**Definition 9 (timed interface automata)** A timed interface automaton (or TIA) is a tuple \( A = (Q_A, q^{in}_A, \mathcal{X}_A, Acts^I_A, Acts^O_A, Inv^I_A, Inv^O_A, \rho_A) \) consisting of the following components.

- \( Q_A \) is a finite set of locations.
- \( q^{in}_A \in Q_A \) is the initial location.
- \( \mathcal{X}_A \) is a finite set of clocks.
- \( Acts^I_A \) and \( Acts^O_A \) are finite and disjoint sets of input and output actions, respectively. Let \( Acts_A = Acts^I_A \cup Acts^O_A \) denote the set of all actions of \( A \).
- \( Inv^I_A : Q_A \mapsto \Xi[\mathcal{X}_A] \) maps each location of \( A \) to its input invariant.
- \( Inv^O_A : Q_A \mapsto \Xi[\mathcal{X}_A] \) maps each location of \( A \) to its output invariant.
\( \rho_A \subseteq Q_A \times \mathcal{X}[X_A] \times \text{Acts}_A \times 2^{X_A} \times Q_A \) is the transition relation. For \((q, g, a, r, q') \in \rho_A\), the locations \(q\) and \(q'\) are the source and destination of the transition, \(g \in \mathcal{X}[X_A]\) is a guard on the clock values that specifies when the transition can be taken, \(a \in \text{Acts}_A\) is an action labeling the transition, and \(r \subseteq X_A\) is a set of clocks that are reset by the transition. We require the transition relation to be deterministic: for all \(q \in Q_A\) and \(a \in \text{Acts}_A\), there is at most one tuple of the form \((q, g, a, r, q')\) with \((q, g, a, r, q') \in \rho_A\).

A valuation over a set \(X\) of clock variables is a function \(v: X \mapsto \mathbb{T}\). We write \(0_X\) for the valuation that assigns 0 to all clocks in \(X\), and \(\mathcal{V}(X)\) for the set of all valuations over \(X\). Given a valuation \(v \in \mathcal{V}(X)\), we write \(v + \Delta\) for the valuation defined by \((v + \Delta)(x) = v(x) + \Delta\) for all \(x \in X\). Given a set \(r \subseteq X\) of clocks, we write \(v[r := 0]\) for the valuation that maps \(x\) to 0 if \(x \in r\), and otherwise to \(v(x)\). Given a clock condition \(\varphi \in \mathcal{X}[X]\), we write \(v \models \varphi\) if \(\varphi\) is true under the valuation \(v\). For \(r \subseteq X\), we write \(\varphi[r := 0]\) for the condition obtained from \(\varphi\) by replacing every \(x \in r\) by 0; obviously, \(v[r := 0]\) \(\models \varphi\) iff \(v \models \varphi[r := 0]\).

**Definition 10 (timed interfaces induced by TIA)** The TIA \(A\) is nonempty if \(0_{X_A} \models \text{Inv}^A_f(q_A^{\text{init}}) \land \text{Inv}^A_o(q_A^{\text{init}})\). A nonempty TIA \(A\) induces a timed interface \(P = [A]\) that has the state set \(S_P = \{p, v\} \mid p \in Q_A, v \in \mathcal{V}(X_A)\) and the initial state \(s_P^{\text{init}} = (q_A^{\text{init}}, 0_{X_A})\). The actions are \(\text{Acts}_P^f = \text{Acts}_A^f\) and \(\text{Acts}_P^o = \text{Acts}_A^o\).

For \(\gamma \in \{I, O\}\) the transition relations of \(P\) are defined by \((p, v) \xrightarrow{a, r} (p', v')\) \(\in p_P^\gamma\) if either (1) \(a \in T\), \(p = p'\), \(v' = v + \alpha\), and for all \(0 \leq \Delta' \leq \alpha\), we have \(v + \Delta' \models \text{Inv}^A_\gamma(p)\); or (2) \(a \in \text{Acts}_A\), and there is a tuple \((p, g, a, r, p') \in \rho_A\) with \(v \models \text{Inv}^A_\gamma(p) \land g, v' = v[r := 0]\), and \(v' \models \text{Inv}^A_\gamma(p')\).

The TIA \(A\) is well-formed if it is nonempty and the corresponding timed interface \([A]\) is well-formed.

**4.1 Product and composition of timed interface automata**

Two TIAs \(A\) and \(B\) are composable if \(\text{Acts}_A^O \cap \text{Acts}_B^O = \emptyset\) and \(X_A \cap X_B = \emptyset\); their shared actions are \(\text{shared}(A, B) = \text{Acts}_A \cap \text{Acts}_B\).

**Definition 11 (product)** For two composable TIAs \(A_1\) and \(A_2\), the product \(A_1 \otimes A_2\) is the TIA that consists of the following components.

- \(Q_{A_1 \otimes A_2} = Q_{A_1} \times Q_{A_2}\), and \(q_{A_1 \otimes A_2}^{\text{init}} = (q_{A_1}^{\text{init}}, q_{A_2}^{\text{init}})\).
- \(X_{A_1 \otimes A_2} = X_{A_1} \cup X_{A_2}\).
- \(\text{Acts}_{A_1 \otimes A_2}^P = \text{Acts}_{A_1}^f \cup \text{Acts}_{A_2}^f \setminus \text{shared}(A_1, A_2)\), and \(\text{Acts}_{A_1 \otimes A_2}^O = \text{Acts}_{A_1}^o \cup \text{Acts}_{A_2}^o\).
- \(\text{Inv}_{A_1 \otimes A_2}^f(p, q) = \text{Inv}_{A_1}^f(p) \land \text{Inv}_{A_2}^f(q)\) and \(\text{Inv}_{A_1 \otimes A_2}^o(p, q) = \text{Inv}_{A_1}^o(p) \land \text{Inv}_{A_2}^o(q)\).
- \(\rho_{A_1 \otimes A_2}\) is the set of transitions \(\{(q_1, q_2), g_1 \land g_2, a, r_1 \cup r_2, (q'_1, q'_2)\}\) such that, for \(i = 1, 2\): if \(a \in \text{Acts}_{A_i}\), then \((q_i, g_i, a, r_i, q'_i)\) is a transition in \(\rho_{A_i}\); otherwise \(q_i = q'_i, g_i = \text{true}\), and \(r_i = \emptyset\).
Theorem 1  For nonempty and composable TIA s A and B, we have $[A \otimes B] = [A] \otimes [B]$.

A location labeling for a TIA $A$ is a function $\xi: Q_A \rightarrow \Xi[X_A]$ that associates with each location $p$ of $A$ a condition $\xi(p)$ over the clocks in $X_A$. The location labeling $\xi$ defines the state set $[\xi]_A = \{ (p, v) \in S_{[A]} \mid v = \xi(p) \}$ of the corresponding timed interface. We denote by $I\text{-live}_A(\xi)$ the location labeling that defines the set $[I\text{-live}_A(\xi)]_A$ of I-live states in $[\xi]_A$. By computing $[I\text{-live}_A(\xi)]_A$ as the solution of a game on the region graph (see Section 4.2), we will see that the I-live states are indeed definable by clock conditions. To define the composition on TIA s, we also need the following enabling conditions. For $\gamma \in \{ I, O \}$, a location $q \in Q_A$, and an action $a \in \text{Acts}_A^\gamma$, let $\text{enab}^\gamma_A(q, a)$ be $\text{Inv}^\gamma_A(q) \land g \land \text{Inv}^\gamma_B(q') \land [r := 0]$ if there is a transition $(q, g, a, r, q') \in \rho_A$; otherwise let $\text{enab}^\gamma_A(q, a)$ be false. Given two composable TIA s $A$ and $B$, the states of $[A \otimes B]$ that are not immediate error states can be defined by the location labeling $ok_{A \otimes B}$ that associates with each product location $(p, q) \in Q_{A \otimes B}$ the clock condition

$$\bigwedge_{a \in \text{Acts}_B^I \cap \text{Acts}_B^O} (\text{enab}_B^I(q, a) \land \text{enab}_B^O(q, a)) \land \bigwedge_{a \in \text{Acts}_A^I \cap \text{Acts}_A^O} (\text{enab}_A^I(p, a) \land \text{enab}_A^O(p, a)).$$

Definition 12 (compatibility and composition) Two well-formed and composable TIA s $A$ and $B$ are compatible if the corresponding timed interface $[A]$ and $[B]$ are compatible. The composition $A || B$ is obtained from the product $A \otimes B$ by replacing the input invariants $\text{Inv}^I_A$ with the location labeling $I\text{-live}_{A \otimes B}(ok_{A \otimes B})$.

Theorem 2  Two well-formed and composable TIA s $A$ and $B$ are compatible iff the composition $A || B$ is nonempty. Moreover, if $A$ and $B$ are compatible, then $[A || B] = [A][[B]]$, and $A || B$ is well-formed.

The following theorem states that composition is associative up to the equivalence $\equiv$, which for TIA s $A$, $B$, and $C$ is defined by $A \equiv B$ if $\rho^I_{[A]} \cap \text{Reach}([A] \times \text{Acts}_A \times S_{[A]}) = \rho^I_{[B]} \cap \text{Reach}([B] \times \text{Acts}_B \times S_{[B]})$ and all other components of $[[A]]$ and $[[B]]$ are the same. Here, $\text{Reach}(\mathcal{P})$ denotes the set of reachable states of the timed interface $\mathcal{P}$.

Theorem 3  If $A$, $B$, and $C$ are well-formed and pairwise composable TIA s, then $(A || B) || C \equiv A || (B || C)$.

4.2 Algorithms for composition and well-formedness checking

Live states. Before presenting the algorithms for checking well-formedness and computing composition, we show that, given a location labeling $\xi$, we can compute the labeling $I\text{-live}_A(\xi)$ that defines the set of states in $A$ where $I$ can win the game on $A$ with goal $(\text{t.div} \lor \text{blame}^O) \land \square[\xi]_A$. 

13
To use existing algorithms, we first transform this game into an equivalent one with an \( \omega \)-regular goal [Tho90]. Consider the TIA \( \text{Tick}^O \) shown on the right, where the action \( \text{tick} \) and the clock \( x \) are fresh. Thus, \( \text{Tick}^O \) observes the progress of time, and visits location \( q_1 \) every time unit. In particular, time diverges iff \( q_1 \) is visited infinitely often. Hence, Input can win in \([A]\) the game with goal \((t \_ \text{div} \lor \text{blame}^O) \land \square [\xi]_A \) iff Input can win in \([A \otimes \text{Tick}^O]\) the game with goal \((\text{blame}^O \lor \square \otimes q_1) \land \square [\xi]_A \). On the enlarged state space \( S_{[A \otimes \text{Tick}^O]} \times \{I, O\} \), where the states record the blame, this latter goal can be rewritten as the \( \omega \)-regular goal \( \varphi_I \): \((\square \otimes (bl = O) \lor \square \otimes q_1) \land \square [\xi]_A \).

The game with goal \( \varphi_I \) can be solved using the algorithms of [EJ91, dAHM01], which use a controllable predecessor operator \( I\text{Pre} \). Given a timed interface \( P \) and a set \( U \) of states, \( I\text{Pre}_P(U) \) yields all states in which Input can in one move force the game into \( U \). Formally, \( I\text{Pre}_P(U) \) contains all states \( s \in S_P \) such that \( \forall \alpha_O \in I_{P}^O(s), \exists \alpha_I \in I_{P}^I(s), \delta_P(s, \alpha_I, \alpha_O) \subseteq U \). The set \( \text{win}_I \) of states in \( S_{[A \otimes \text{Tick}^O]} \times \{I, O\} \) where Input can win the game with goal \( \varphi_I \) can be characterized by [EJ91]

\[
\text{win}_I = \nu Z. \mu Y. \nu X. \left[ \xi \land \left( (q_0 \land bl = O \land I\text{Pre}_P(X)) \lor (q_0 \land bl = I \land I\text{Pre}_P(Y)) \lor (q_1 \land I\text{Pre}_P(Z)) \right) \right].
\]

for \( P = [A \otimes \text{Tick}^O] \). One can define the region graph of a TIA as for timed automata [AD94]. Since the operation \( I\text{Pre} \) is computable on the region graph [MPS95], the expression above suggests a symbolic fixpoint algorithm. The result \( \text{win}_I \) can be expressed as a location labeling \( \zeta \) on \( A \). We then obtain the desired labeling \( I\text{-live}_A(\xi) \) by letting \( I\text{-live}_A(\xi)(p) = \exists x. \exists bl. (\zeta(p, q_0) \lor \zeta(p, q_1)) \).

**Well-formedness.** The following theorem shows that, to check the well-formedness of a TIA \( A \), we need to compute the location labelings \( \text{Reach}(\xi), I\text{-live}_A(\text{True}_A), \) and \( O\text{-live}_A(\text{True}_A) \). Here, \( \text{Reach}(\xi) \) is the location labeling that defines the set \( \llbracket \text{Reach}(\xi) \rrbracket_A \) of reachable states of \( [A] \), and \( O\text{-live}_A(\xi) \) is the labeling that defines the set of O-live states in \( [\xi]_A \), and \( \text{True}_A \) denotes the labeling that assigns \text{true} to each location. The set \( \llbracket \text{Reach}(\xi) \rrbracket_A \) is definable from clock conditions, because it can be computed on the region graph in the same way as the reachable states of a timed automaton can be computed [AD94]. Since \( [O\text{-live}_A(\xi)]_A \), for a location labeling \( \xi \), can be computed similarly to \( [I\text{-live}_A(\xi)]_A \), it is also definable by clock conditions.

**Theorem 4** A TIA \( A \) is well-formed iff for all locations \( p \in Q_A \) the implication \( \text{Reach}(\xi)(p) \rightarrow (I\text{-live}_A(\text{True}_A)(p) \land O\text{-live}_A(\text{True}_A)(p)) \) is valid.

**Composition.** The composition of two well-formed and composable TIAs \( A \) and \( B \) can be obtained from their product by replacing the input invariants \( \text{Inv}_A \) with the location labeling \( I\text{-live}_{A \otimes B}(ok_{A \otimes B}) \) (Definition 12). Their compatibility can be decided by checking whether their composition is empty (Theorem 2).
References


