Compliance and Subtyping in Timed Session Types

Massimo Bartoletti, Tiziana Cimoli, Maurizio Murgia, Alessandro Sebastian Podda, Livio Pompianu

Compliance and subtyping in timed session types

Massimo Bartoletti, Tiziana Cimoli, Maurizio Murgia,
Alessandro Sebastian Podda, and Livio Pompianu

Università degli Studi di Cagliari, Italy

Abstract. We propose an extension of binary session types, to formalise timed communication protocols between two participants at the endpoints of a session. We introduce a decidable compliance relation, which generalises to the timed setting the usual progress-based notion of compliance between untimed session types. We then show a sound and complete technique to decide when a timed session type admits a compliant one, and if so, to construct the least session type compliant with a given one, according to the subtyping preorder induced by compliance. Decidability of subtyping follows from these results. We exploit our theory to design and implement a message-oriented middleware, where distributed modules with compliant protocols can be dynamically composed, and their communications monitored, so to guarantee safe interactions.

1 Introduction

Session types are formal descriptions of interaction protocols involving two or more participants over a network [18,23]. They can be used to specify the behavioural interface of a service or a component, and to statically check through a (session-)type system that this conforms to its implementation, so enabling compositional verification of distributed applications. Session types support formal definitions of compatibility or *compliance* (when two or more session types, composed together, behave correctly), and of substitutability or *subtyping* (when a service can be safely replaced by another one, while preserving the interaction capabilities with the context). Since these notions are often decidable and computationally tractable (for synchronous session types), or safely approximable (for asynchronous ones), session typing is becoming a particularly attractive approach to the problem of correctly designing distributed applications. This is witnessed by a steady flow of foundational studies [16,10,15] and of tools [12,24] based on them in the last few years.

In the simplest setting, session types are terms of a process algebra featuring a selection construct (an *internal choice* among a set of branches), a branching construct (an *external choice* offered to the environment), and recursion. In this basic form, session types cannot faithfully capture a natural and relevant aspect of interaction protocols, i.e., the timing constraints among the communication actions. While formal methods for time have been studied for at least a couple of decades, they have approached the realm of session types very recently [9,22].

However, these approaches introduce time into an already sophisticated framework, featuring multiparty session types with asynchronous communication (via unbounded buffers). While on the one hand this has the advantage of extending to the timed setting type techniques which enable compositional verification [19], on the other hand it seems that some of the key notions of the untimed setting (e.g., compliance, duality) have not been explored yet in the timed case.

We think that studying timed session types in a basic setting (synchronous communication between two endpoints, as in the seminal untimed version) is worthy of attention. From a theoretical point of view, the objective is to lift to the timed case some decidability results, like those of compliance and subtyping. Some intriguing problems arise: unlike in the untimed case, a timed session type not always admits a compliant; hence, besides deciding if two session types *are compliant*, it becomes a relevant problem whether a session type *has a compliant*. From a more practical perspective, decision procedures for timed session types, like those for compliance and for dynamic verification, enable the implementation of programming tools and infrastructures for the development of safe communication-oriented distributed applications.

Contributions. In this paper we introduce a theory of binary timed session types (TSTs), and we explore its viability as a foundation for programming tools to leverage the complexity of developing distributed applications.

We start in Section 2 by giving the syntax and semantics of TSTs. E.g., we describe as the following TST the contract of a service taking as input a zip code, and then either providing as output the current weather, or aborting:

$$p \;=\; \texttt{?zip}\{x\}.\,(\texttt{!weather}\{5 < x < 10\} \oplus \texttt{!abort}\{x < 1\})$$

The prefix $\texttt{?zip}\{x\}$ states that the service can receive a `zip` code, and then reset a clock x. The continuation is an *internal choice* between two outputs: either the service sends `weather` in a time window of $(5, 10)$ time units, or it will `abort` the protocol within 1 time unit.

The semantics of TSTs is a conservative extension of the synchronous semantics of untimed session types [4], adding *clock valuations* to associate each clock with a positive real. We also extend to the timed setting the standard semantic notion of *compliance*, which relates two session types whenever they enjoy progress until reaching success. For instance, p above is *not* compliant with:

$$q \;=\; \texttt{!zip}\{y\}.\,(\texttt{?weather}\{y < 7\} + \texttt{?abort}\{y < 5\})$$

because q is available to receive `weather` until 7 time units since it has sent the `zip` code, while p can choose to send `weather` until 10 time units (note that p and q, cleaned from all time annotations, are compliant in the untimed setting).

Despite the semantics of TSTs being infinite-state (while it is finite-state in the untimed case), we develop a sound and complete decision procedure for verifying compliance (Theorem 1). To do that, we reduce this problem to that of model-checking deadlock freedom in timed automata [2], which is decidable, and we implement our technique using the Uppaal model checker [7].

Another difference from the untimed case is that not every TST admits a compliant (while in the untimed case, a session type is always compliant to its syntactic dual). For instance, consider the client contract:

$$q' \;=\; !\mathtt{zip}\{y < 10\}.\,(?\mathtt{weather}\{y < 7\} + ?\mathtt{abort}\{y < 5\})$$

No service can be compliant with q', because if q' sends the `zip` code, e.g., at time 8, one cannot send `weather` or `abort` in the given time constraints. We develop a procedure to detect whether a TST admits a compliant. This takes the form of a kind system which associates, to each p, a set of clock valuations under which p admits a compliant. The kind system is sound and complete (Theorems 4 and 5), and kind inference is decidable (Theorem 3), so summing up we have a (sound and complete) decision procedure for the existence of compliant. When p admits a compliant, by exploiting the kind system we can construct the *greatest* TST compliant with p (Theorem 7), according to the semantic subtyping relation [4]. Decidability of subtyping follows from that of compliance and kind inference. This provides us with an effective way of checking whether a service with type p can be replaced by one with a subtype p' of p, guaranteeing that all the services which interacted correctly with the old one will do the same with the new one.

In Section 4 we address the problem of dynamically monitoring interactions regulated by TSTs. To do that, we will provide TSTs with a *monitoring semantics*, which detects when a participant is not respecting its TST. This semantics enjoys some desirable properties: it is deterministic, and it guarantees that in each state of an interaction, either we have reached success, or someone is in charge of a move, or not respecting its TST. We then exploit all the theoretical results discussed above, to discuss the design and implementation of a message-oriented middleware which uses TSTs to enable and regulate the interaction of distributed services. This infrastructure pursues the bottom-up approach to service composition: it allows services to advertise contracts (in the form of TSTs); all the advertised TSTs are collected by a *broker*, which finds pairs of compliant TSTs, and creates *sessions* between the respective services. These can then start interacting, by doing the actions prescribed by their TSTs (or even by choosing not to do so). In a system of honest services, compliance between TSTs ensures progress of the whole system; in any case, dynamic verification of all the exchanged messages guarantees safe executions.

Due to space constraints, the proofs of our statements, additional examples, as well as some tools related to the middleware, are available in [5].

2 Timed session types

We introduce binary timed session types (TSTs), by giving their syntax and semantics, and by defining a compliance relation between them. The main result of this section is Theorem 1, which states that compliance is decidable.

Syntax. Let $\mathtt{A}$ be a set of *actions*, ranged over by $\mathtt{a}, \mathtt{b}, \ldots$. We denote with $\mathtt{A}^!$ the set $\{!\mathtt{a} \mid \mathtt{a} \in \mathtt{A}\}$ of *output actions*, with $\mathtt{A}^?$ the set $\{?\mathtt{a} \mid \mathtt{a} \in \mathtt{A}\}$ of *input actions*, and with $\mathtt{L} = \mathtt{A}^! \cup \mathtt{A}^?$ the set of *branch labels*, ranged over by $\ell, \ell', \ldots$.

We use $\delta, \delta', \ldots$ to range over the set $\mathbb{R}_{\geq 0}$ of positive real numbers including zero, and $d, d', \ldots$ to range over $\mathbb{N}$. Let $\mathbb{C}$ be a set of *clocks*, namely variables in $\mathbb{R}_{\geq 0}$, ranged over by $t, t', \ldots$. We use $R, T, \ldots \subseteq \mathbb{C}$ to range over sets of clocks.

Definition 1 (Guards). *The set $\mathcal{G}_{\mathbb{C}}$ of* guards *over clocks $\mathbb{C}$ is defined as:*

$$g \;::=\; \mathtt{true} \;\mid\; \neg g \;\mid\; g \wedge g \;\mid\; t \circ d \;\mid\; t - t' \circ d \quad (\textit{where } \circ \in \{<, \leq, =, \geq, >\})$$

Definition 2 below introduces the syntax of TSTs. A TST p models the behaviour of a single participant involved in an interaction. TSTs are terms of a process algebra featuring the *success* state $\mathbf{1}$, *internal choice* $\bigoplus_{i \in I} \mathtt{!a}_i\{g_i, R_i\}\,.\,p_i$, *external choice* $\sum_{i \in I} \mathtt{?a}_i\{g_i, R_i\}\,.\,p_i$, and recursion $\mathrm{rec}\, X\,.\,p$.

To give some intuition, we consider two participants, Alice (A) and Bob (B), which want to interact. Alice advertises an internal choice $\bigoplus_i \mathtt{!a}_i\{g_i, R_i\}\,.\,p_i$ when she wants to do one of the outputs $\mathtt{!a}_i$ in a time window where g_i is true; further, the clocks in R_i will be reset after the output is performed. The meaning of an external choice $\sum_i \mathtt{?a}_i\{g_i, R_i\}\,.\,q_i$ (advertised, say, by B) is somehow dual: B is saying that he is available to receive each message $\mathtt{a}_i$ in *any instant* within the time window defined by g_i (and the clocks in R_i will be reset after the input).

Definition 2 (Timed session types). Timed session types $p, q, \ldots$ *are terms of the following grammar:*

$$p \;::=\; \mathbf{1} \;\mid\; \bigoplus_{i \in I} \mathtt{!a}_i\{g_i, R_i\}\,.\,p_i \;\mid\; \sum_{i \in I} \mathtt{?a}_i\{g_i, R_i\}\,.\,p_i \;\mid\; \mathrm{rec}\, X\,.\,p \;\mid\; X$$

where (i) the set I is finite and non-empty, (ii) the actions in internal/external choices are pairwise distinct, (iii) recursion is guarded. Unless stated otherwise, we consider TSTs up-to unfolding of recursion. A TST is closed *when it has no recursion variables. If $q = \bigoplus_{i \in I} \mathtt{!a}_i\{g_i, R_i\}\,.\,p_i$ and $0 \notin I$, we write $\mathtt{!a}_0.p_0 \oplus q$ for $\bigoplus_{i \in I \cup \{0\}} \mathtt{!a}_i\{g_i, R_i\}\,.\,p_i$ (the same for external choices). True guards, empty resets, and trailing occurrences of the success state can be omitted.*

Example 1. Along the lines of PayPal User Agreement [1], we specify the protection policy for buyers of a simple on-line payment platform, called PayNow (see [5] for the full version). PayNow helps customers in on-line purchasing, providing protection against misbehaviours. In case a buyer has not received what he has paid for, he can open a dispute within 180 days from the date the buyer made the payment. After opening of the dispute, the buyer and the seller may try to come to an agreement. If this is not the case, within 20 days, the buyer can escalate the dispute to a claim. However, the buyer must wait at least 7 days from the date of payment to escalate a dispute. Upon not reaching an agreement, if still the buyer does not escalate the dispute to a claim within 20 days, the dispute is considered aborted. During a claim procedure, PayNow will ask the buyer to provide documentation to certify the payment, within 3 days of the date the dispute was escalated to a claim. After that, the payment will be refunded

within 7 days. The contract of PayNow is described by the following TST p:

$$\begin{aligned} p &= \texttt{?pay}\{t_{pay}\}.\,(\texttt{?ok} + \texttt{?dispute}\{t_{pay} < 180, t_d\}.\,p') && \text{where} \\ p' &= \texttt{?ok}\{t_d < 20\} + \\ &\quad \texttt{?claim}\{t_d < 20 \wedge t_{pay} > 7, t_c\}.\texttt{?rcpt}\{t_c < 3, t_c\}.\texttt{!refund}\{t_c < 7\} + \\ &\quad \texttt{?abort} \end{aligned}$$

Semantics. To define the behaviour of TSTs we use *clock valuations*, which associate each clock with its value. The state of the interaction between two TSTs is described by a *configuration* $(p, \nu) \mid (q, \eta)$, where the clock valuations ν and η record (keeping the same pace) the time of the clocks in p and q, respectively. The dynamics of the interaction is formalised as a transition relation between configurations (Definition 5). This relation describes all and only the *correct* interactions: for instance, we do not allow time passing to make unsatisfiable all the guards in an internal choice, since doing so would prevent a participant from respecting her protocol. In Section 4 we will study another semantics of TSTs, which can also describe the behaviour of dishonest participants who do not respect their protocols.

We denote with $\mathbb{V} = \mathbb{C} \to \mathbb{R}_{\geq 0}$ the set of clock valuations (ranged over by $\nu, \eta, \ldots$), and with ν_0 the valuation mapping each clock to zero. We write $\nu + \delta$ for the valuation which increases ν by δ, i.e., $(\nu + \delta)(t) = \nu(t) + \delta$ for all $t \in \mathbb{C}$. For a set $R \subseteq \mathbb{C}$, we write $\nu[R]$ for the *reset* of the clocks in R, i.e., $\nu[R](t) = 0$ if $t \in R$, and $\nu[R](t) = \nu(t)$ otherwise.

Definition 3 (Semantics of guards). *For all guards g, we define the set of clock valuations $[\![g]\!]$ inductively as follows, where $\circ \in \{<, \leq, =, \geq, >\}$:*

$$[\![\texttt{true}]\!] = \mathbb{V} \qquad [\![\neg g]\!] = \mathbb{V} \setminus [\![g]\!] \qquad [\![g_1 \wedge g_2]\!] = [\![g_1]\!] \cap [\![g_2]\!]$$
$$[\![t \circ d]\!] = \{\nu \mid \nu(t) \circ d\} \qquad [\![t - t' \circ d]\!] = \{\nu \mid \nu(t) - \nu(t') \circ d\}$$

Before defining the semantics of TSTs, we recall from [8] some basic operations on *sets* of clock valuations (ranged over by $\mathcal{K}, \mathcal{K}', \ldots \subseteq \mathbb{V}$).

Definition 4 (Past and inverse reset). *For all sets $\mathcal{K}$ of clock valuations, the set of clock valuations $\downarrow \mathcal{K}$ (the* past *of $\mathcal{K}$) and $\mathcal{K}[T]^{-1}$ (the inverse reset of $\mathcal{K}$) are defined as: $\downarrow \mathcal{K} = \{\nu \mid \exists \delta \geq 0 : \nu + \delta \in \mathcal{K}\}$, $\mathcal{K}[T]^{-1} = \{\nu \mid \nu[T] \in \mathcal{K}\}$.*

Definition 5 (Semantics of TSTs). *A* configuration *is a term of the form $(p, \nu) \mid (q, \eta)$, where p, q are TSTs extended with* committed choices $[\texttt{!a}\{g, R\}]\,p$. *The semantics of TSTs is defined as a labelled relation $\to$ over configurations, whose labels are either silent actions τ, delays δ, or branch labels.*

We now comment the rules in Figure 1. The first four rules are auxiliary, as they describe the behaviour of a TST in isolation. Rule $[\oplus]$ allows a TST to commit to the branch $\texttt{!a}$ of her internal choice, provided that the corresponding

$$(\texttt{!a}\{g, R\}.\, p \oplus p',\ \nu) \xrightarrow{\tau} ([\texttt{!a}\{g, R\}]\, p,\ \nu) \qquad \text{if } \nu \in [\![g]\!] \qquad [\oplus]$$

$$([\texttt{!a}\{g, R\}]\, p,\ \nu) \xrightarrow{\texttt{!a}} (p,\ \nu[R]) \qquad [!]$$

$$(\texttt{?a}\{g, R\}.\, p + p',\ \nu) \xrightarrow{\texttt{?a}} (p,\ \nu[R]) \qquad \text{if } \nu \in [\![g]\!] \qquad [?]$$

$$(p,\ \nu) \xrightarrow{\delta} (p,\ \nu + \delta) \qquad \text{if } \delta > 0 \ \wedge\ \nu + \delta \in \texttt{rdy}(p) \qquad [\textsc{Del}]$$

$$\frac{(p,\nu) \xrightarrow{\tau} (p',\nu')}{(p,\nu) \mid (q,\eta) \xrightarrow{\tau} (p',\nu') \mid (q,\eta)}\ [\text{S-}\oplus] \qquad \frac{(p,\nu) \xrightarrow{\delta} (p,\nu') \quad (q,\eta) \xrightarrow{\delta} (q,\eta')}{(p,\nu) \mid (q,\eta) \xrightarrow{\delta} (p,\nu') \mid (q,\eta')}\ [\text{S-}\textsc{Del}]$$

$$\frac{(p,\nu) \xrightarrow{\texttt{!a}} (p',\nu') \quad (q,\eta) \xrightarrow{\texttt{?a}} (q',\eta')}{(p,\nu) \mid (q,\eta) \xrightarrow{\tau} (p',\nu') \mid (q',\eta')}\ [\text{S-}\tau]$$

$$\texttt{rdy}(\textstyle\bigoplus \texttt{!a}_i\{g_i, R_i\}.\, p_i) = {\downarrow}\bigcup [\![g_i]\!] \qquad \texttt{rdy}(\textstyle\sum \cdots) = \texttt{rdy}(\mathbf{1}) = \mathbb{V} \qquad \texttt{rdy}([\texttt{!a}\{g, R\}]\, p) = \emptyset$$

Fig. 1. Semantics of timed session types (symmetric rules omitted).

guard is satisfied in the clock valuation ν. This results in the term $[\texttt{!a}\{g, R\}]\, p$, which represents the fact that the endpoint has committed to branch `!a` in a specific time instant: actually, it can only fire `!a` through rule [!] (which also resets the clocks in R), while time cannot pass. Rule [?] allows an external choice to fire any of its input actions whose guard is satisfied. Rule [Del] allows time to pass; this is always possible for external choices and success term, while for an internal choice we require that at least one of the guards remains satisfiable; this is obtained through the function `rdy` in Figure 1. The last three rules deal with configurations of two TSTs. Rule [S-$\oplus$] allows a TSTs to commit in an internal choice. Rule [S-τ] is the standard synchronisation rule *à la* CCS; note that B is assumed to read a message as soon as it is sent, so A never blocks on internal choices. Rule [S-Del] allows time to pass, equally for both endpoints.

Example 2. Let $p = \texttt{!a} \oplus \texttt{!b}\{t > 2\}$, let $q = \texttt{?b}\{t > 5\}$, and consider the following computations:

$$\begin{aligned}(p,\nu_0) \mid (q,\eta_0) &\xrightarrow{7}\xrightarrow{\tau} ([\texttt{!b}\{t > 2\}]\,, \nu_0 + 7) \mid (q, \eta_0 + 7) \\ &\xrightarrow{\tau} (\mathbf{1}, \nu_0 + 7) \mid (\mathbf{1}, \eta_0 + 7) \qquad (1)\end{aligned}$$

$$(p,\nu_0) \mid (q,\eta_0) \xrightarrow{\delta}\xrightarrow{\tau} ([\texttt{!a}]\,, \nu_0 + \delta) \mid (q, \eta_0 + \delta) \qquad (2)$$

$$(p,\nu_0) \mid (q,\eta_0) \xrightarrow{3}\xrightarrow{\tau} ([\texttt{!b}\{t > 2\}]\,, \nu_0 + 3) \mid (q, \eta_0 + 3) \qquad (3)$$

The computation in (1) reaches success, while the other two computations reach the deadlock state. In (2), p commits to the choice `!a` after some delay δ; at this point, time cannot pass (because the leftmost endpoint is a committed choice), and no synchronisation is possible (because the other endpoint is not offering `?a`). In (3), p commits to `!b` after 3 time units; here, the rightmost endpoint would

offer $\texttt{?b}$, — but not in the time chosen by the leftmost endpoint. Note that, were we allowing time to pass in committed choices, then we would have obtained e.g. that $(\texttt{!b}\{t > 2\}, \nu_0) \mid (q, \eta_0)$ never reaches deadlock — contradicting our intuition that these endpoints should not be considered compliant.

Compliance. We now extend to the timed setting the standard progress-based compliance between (untimed) session types [21,11,4]. If p is compliant with q, then whenever an interaction between p and q becomes stuck, it means that both participants have reached the success state. Intuitively, when two TSTs are compliant and participants behave honestly (according to their TSTs), then the interaction will progress, until both of them reach the success state.

Definition 6 (Compliance). *We say that* $(p, \nu) \mid (q, \eta)$ *is* deadlock *whenever* (i) *it is not the case that both* p *and* q *are* $\mathbf{1}$*, and* (ii) *there is no* δ *such that* $(p, \nu + \delta) \mid (q, \eta + \delta) \xrightarrow{\tau}$*. We then write* $(p, \nu) \bowtie (q, \eta)$ *whenever:*

$$(p, \nu) \mid (q, \eta) \rightarrow^* (p', \nu') \mid (q', \eta') \quad \textit{implies} \quad (p', \nu') \mid (q', \eta') \textit{ not deadlock}$$

We say that p *and* q *are* compliant *whenever* $(p, \nu_0) \bowtie (q, \eta_0)$ *(in short,* $p \bowtie q$*).*

Example 3. Let $p = \texttt{?a}\{t < 5\}.\texttt{!b}\{t < 3\}$. We have that p is compliant with $q = \texttt{!a}\{t < 2\}.\texttt{?b}\{t < 3\}$, but it is not compliant with $q' = \texttt{!a}\{t < 5\}.\texttt{?b}\{t < 3\}$.

Example 4. Consider a customer of PayNow (see Example 1) who is willing to wait 10 days to receive the item she has paid for, but after that she will open a claim. Further, she will instantly provide PayNow with any documentation required. The customer contract is described by the following TST, which is compliant with PayNow's contract p in Example 1:

$$\texttt{!pay}\{t_{pay}\}.(\texttt{!ok}\{t_{pay} < 10\} \oplus \texttt{!dispute}\{t_{pay} = 10\}.\texttt{!claim}\{t_{pay} = 10\}.\texttt{!rcpt}\{t_{pay} = 10\}.\texttt{?refund})$$

Compliance between TSTs is somehow more liberal than the untimed notion, as it can relate terms which, when cleaned from all the time annotations, would not be compliant in the untimed case. The following example shows e.g., that a recursive internal choice can be compliant with a non-recursive external choice — which can never happen in untimed session types.

Example 5. Consider the TSTs $p = \mathrm{rec}\, X.\,(\texttt{!a} \oplus \texttt{!b}\{x \leq 1\}.\texttt{?c}.X)$, and $q = \texttt{?a} + \texttt{?b}\{y \leq 1\}.\texttt{!c}\{y > 1\}.\texttt{?a}$. We have that $p \bowtie q$. Indeed, if p chooses the output $\texttt{!a}$, then q has the corresponding input, and they both succeed; instead, if p chooses $\texttt{!b}$, then it will read $\texttt{?c}$ when $x > 1$, and so at the next loop it is forced to choose $\texttt{!a}$, since the guard of $\texttt{!b}$ has become unsatisfiable.

Definition 7 and Lemma 1 below coinductively characterise compliance between TSTs, by extending to the timed setting the coinductive compliance for untimed session types in [3]. Intuitively, an internal choice p is compliant with q when (i) q is an external choice, (ii) for each output $\texttt{!a}$ that p can fire after δ time units, there exists a corresponding input $\texttt{?a}$ that q can fire after δ time units, and (iii) their continuations are coinductively compliant. The case where p is an external choice is symmetric.

Definition 7. *We say* $\mathcal{R}$ *is a* coinductive compliance *iff* $(p,\nu)\,\mathcal{R}\,(q,\eta)$ *implies:*

1. $p = \mathbf{1} \iff q = \mathbf{1}$
2. $p = \bigoplus_{i\in I} \mathtt{!a}_i\{g_i, R_i\}\,.\,p_i \implies \nu \in \mathtt{rdy}(p) \;\wedge\; q = \sum_{j\in J} \mathtt{?a}_j\{g_j, R_j\}\,.\,q_j \;\wedge$
   $\forall \delta, i:\ \nu+\delta \in [\![g_i]\!] \implies \exists j : \mathtt{a}_i = \mathtt{a}_j \wedge \eta+\delta \in [\![g_j]\!] \wedge (p_i, \nu+\delta[R_i])\,\mathcal{R}\,(q_j, \eta+\delta[R_j])$
3. $p = \sum_{j\in J} \mathtt{?a}_j\{g_j, R_j\}\,.\,p_j \implies \eta \in \mathtt{rdy}(q) \;\wedge\; q = \bigoplus_{i\in I} \mathtt{!a}_i\{g_i, R_i\}\,.\,q_i \;\wedge$
   $\forall \delta, i:\ \eta+\delta \in [\![g_i]\!] \implies \exists j : \mathtt{a}_i = \mathtt{a}_j \wedge \nu+\delta \in [\![g_j]\!] \wedge (p_j, \nu+\delta[R_j])\,\mathcal{R}\,(q_i, \eta+\delta[R_i])$

Lemma 1. $p \bowtie q \iff \exists\, \mathcal{R}$ *coinductive compliance* $:\ (p,\nu_0)\,\mathcal{R}\,(q,\eta_0)$

The following theorem establishes decidability of compliance. To prove it, we reduce the problem of checking $p \bowtie q$ to that of model-checking deadlock freedom in a network of timed automata constructed from p and q (see [5] for details).

Theorem 1. *Compliance between TSTs is decidable.*

3 On duality and subtyping

The dual of an untimed session type is computed by simply swapping internal choices with external ones (and inputs with outputs) [10]. A naïve attempt to extend this construction to TSTs can be to swap internal with external choices, as in the untimed case, and leave guards and resets unchanged. This construction does not work as expected, as shown by the following example.

Example 6. Consider the following TSTs:

$$p_1 = \mathtt{!a}\{x \le 2\}.\,\mathtt{!b}\{x \le 1\} \qquad p_2 = \mathtt{!a}\{x \le 2\} \oplus \mathtt{!b}\{x \le 1\}.\,\mathtt{?a}\{x \le 0\}$$
$$p_3 = \mathrm{rec}\,X\,.\,\mathtt{?a}\{x \le 1 \wedge y \le 1\}.\,\mathtt{!a}\{x \le 1, \{x\}\}.\,X$$

The TST p_1 is not compliant with its naïve dual $q_1 = \mathtt{?a}\{x \le 2\}.\,\mathtt{?b}\{x \le 1\}$: even though q_1 can do the input $\mathtt{?a}$ in the required time window, p_1 cannot perform $\mathtt{!b}$ if $\mathtt{!a}$ is performed after 1 time unit. For this very reason, no TST is compliant with p_1. Note instead that $q_1 \bowtie \mathtt{!a}\{x \le 1\}.\,\mathtt{!b}\{x \le 1\}$, which is not its naïve dual. In p_2, a similar deadlock situation occurs if the $\mathtt{!b}$ branch is chosen, and so also p_2 does not admit a compliant. The reason why p_3 does not admit a compliant is more subtle: actually, p_3 can loop until the clock y reaches the value 1; after this point, the guard $y \le 1$ can no longer be satisfied, and then p_3 reaches a deadlock.

As suggested in the above example, the dual construction makes sense only for those TSTs for which a compliant exists. To this purpose, we define a procedure (more precisely, a kind system) which computes the set of clock valuations $\mathcal{K}$ (called *kinds*) such that p admits a compliant TST in all $\nu \in \mathcal{K}$. We then provide a constructive proof of its soundness, by showing a TST q compliant with p, which we call the dual of p.

We now define our kind system for TSTs.

$$\Gamma \vdash \mathbf{1} : \mathbb{V} \ \text{[T-1]} \qquad \frac{\Gamma \vdash p_i : \mathcal{K}_i \qquad \text{for } i \in I}{\Gamma \vdash \sum_{i\in I} \mathtt{?a}_i\{g_i, T_i\}\,.\,p_i : \bigcup_{i\in I} {\downarrow}\left([\![g_i]\!] \cap \mathcal{K}_i[T_i]^{-1}\right)} \ \text{[T-+]}$$

$$\frac{\Gamma \vdash p_i : \mathcal{K}_i \qquad \text{for } i \in I}{\Gamma \vdash \bigoplus_{i\in I} \mathtt{!a}_i\{g_i, T_i\}\,.\,p_i : \left(\bigcup_{i\in I} {\downarrow}[\![g_i]\!]\right) \setminus \left(\bigcup_{i\in I} {\downarrow}([\![g_i]\!] \setminus \mathcal{K}_i[T_i]^{-1})\right)} \ \text{[T-}\oplus\text{]}$$

$$\Gamma, X : \mathcal{K} \vdash X : \mathcal{K} \ \text{[T-Var]} \qquad \frac{\exists \mathcal{K}, \mathcal{K}' : \ \Gamma\{^{\mathcal{K}}/_X\} \vdash p : \mathcal{K}'}{\Gamma \vdash \operatorname{rec} X\,.\,p : \bigcup \{\mathcal{K} \mid \Gamma\{^{\mathcal{K}}/_X\} \vdash p : \mathcal{K}' \wedge \mathcal{K} \subseteq \mathcal{K}'\}} \ \text{[T-Rec]}$$

Fig. 2. Kind system for TSTs.

Definition 8 (Kind system). *Kind judgements $\Gamma \vdash p : \mathcal{K}$ are defined in Figure 2. where Γ is a partial function which associates kinds to recursion variables.*

Rule [T-1] says that the success TST $\mathbf{1}$ admits compliant in every ν: indeed, $\mathbf{1}$ is compliant with itself. The kind of an exernal choice is the union of the kinds of its branches (rule [T-+]), where the kind of a branch is the past of those clock valuations which satisfy both the guard and, after the reset, the kind of their continuation. Internal choices are dealt with by rule [T-⊕], which computes the difference between the union of the past of the guards and a set of error clock valuations. The error clock valuations are those which can satisfy a guard but not the kind of its continuation. Rule [T-Var] is standard. Rule [T-Rec] looks for a kind which is preserved by unfolding of recursion (hence a fixed point). In order to obtain completeness of the kind system we need the greatest fixed point.

Example 7. Recall p_2 from Example 6. We have the following kind derivation:

$$\frac{\vdash \mathbf{1} : \mathbb{V} \qquad \dfrac{\vdash \mathbf{1} : \mathbb{V}}{\vdash \mathtt{!a}\{x \le 0\} : {\downarrow}[\![x \le 0]\!] \cap \mathbb{V} = [\![x \le 0]\!]} \ \text{[T-+]}}{\vdash p_2 : \left({\downarrow}[\![x \le 2]\!] \cup {\downarrow}[\![x \le 1]\!]\right) \setminus \left({\downarrow}[\![x \le 2]\!] \setminus \mathbb{V}\right) \cup {\downarrow}[\![x \le 1]\!] \setminus [\![x \le 0]\!]\big) = \mathcal{K}} \ \text{[T-}\oplus\text{]}$$

where $\mathcal{K} = [\![(x > 1) \wedge (x \le 2)]\!]$. As noted in Example 6, intuitively p_2 has no compliant; this will be asserted by Theorem 5 below, as a consequence of the fact that $\nu_0 \notin \mathcal{K}$. However, since $\mathcal{K}$ is non-empty, Theorem 4 guarantees that there exist q and η such that $(p_2, \nu) \bowtie (q, \eta)$, for all clock valuations $\nu \in \mathcal{K}$.

The following theorem states that every TST is kindable. We stress the fact that being kindable does not imply admitting a compliant. This holds if and only if ν_0 belongs to the kind (see Theorems 4 and 5).

Theorem 2. *For all closed p, there exists some $\mathcal{K}$ such that $\vdash p : \mathcal{K}$.*

The following theorem states that the problem of determining the kind of a TST is decidable. This might seem surprising, as the cardinality of kinds is $2^{2^{\aleph_0}}$. However, the kinds constructed by our inference rules can always be represented syntactically by guards (as in Definition 1) [17].

Theorem 3. *Kind inference is decidable.*

$$
\begin{array}{rll}
\mathrm{co}_\Gamma(\mathbf{1}) = \mathbf{1} & & \\
\mathrm{co}_\Gamma\left(\sum_{i\in I} \mathtt{?a}_i\{g_i, T_i\}\,.\,p_i\right) = \bigoplus_{i\in I} \mathtt{!a}_i\{g_i \wedge \mathcal{K}_i[T_i]^{-1}, T_i\}\,.\,\mathrm{co}_\Gamma(p_i) & \text{if } \Gamma \vdash p_i : \mathcal{K}_i \\
\mathrm{co}_\Gamma\left(\bigoplus_{i\in I} \mathtt{!a}_i\{g_i, T_i\}\,.\,p_i\right) = \sum_{i\in I} \mathtt{?a}_i\{g_i, T_i\}\,.\,\mathrm{co}_\Gamma(p_i) & \\
\mathrm{co}_\Gamma(X) = X & \text{if } \Gamma(X) \text{ defined} \\
\mathrm{co}_\Gamma(\mathrm{rec}\,X\,.\,p) = \mathrm{rec}\,X\,.\,\mathrm{co}_{\Gamma\{\mathcal{K}/X\}}(p) & \text{if } \Gamma \vdash \mathrm{rec}\,X\,.\,p : \mathcal{K}
\end{array}
$$

Fig. 3. Dual of a TST.

We now define the *canonical compliant* of kindable TSTs. Roughly, we turn internal choices into external ones (without changing guards nor resets), and external into internal, changing the guards so that the kind of continuations is preserved. Decidability of this construction follows from that of kind inference.

Definition 9 (Dual). *For all kindable p and kinding environments Γ, we define the TST $\mathrm{co}_\Gamma(p)$ (in short, $\mathrm{co}(p)$ when $\Gamma = \emptyset$) in Figure 3.*

The following theorem states the soundness of the kind system: is particular, if the clock valuation ν_0 belongs to the kind of p, then p admits a compliant.

Theorem 4 (Soundness). *If $\vdash p : \mathcal{K}$ and $\nu \in \mathcal{K}$, then $(p, \nu) \bowtie (\mathrm{co}(p), \nu)$.*

Example 8. Recall the TST $q_1 = \mathtt{?a}\{x \leq 2\}.\,\mathtt{?b}\{x \leq 1\}$ in Example 6. We have:

$$\mathrm{co}(q_1) \;=\; \mathtt{!a}\{x \leq 1\}.\,\mathtt{!b}\{x \leq 1\}$$

Since $\vdash q_1 : \mathcal{K} = [\![x \leq 1]\!]$ and $\nu_0 \in \mathcal{K}$, by Theorem 4 we have that $q_1 \bowtie \mathrm{co}(q_1)$, as anticipated in Example 6.

The following theorem states the kind system is also complete: in particular, if p admits a compliant, then the clock valuation ν_0 belongs to the kind of p.

Theorem 5 (Completeness). *If $\vdash p : \mathcal{K}$ and $\exists q, \eta.\ (p, \nu) \bowtie (q, \eta)$, then $\nu \in \mathcal{K}$.*

Compliance is not transitive, in general (see [5]); however, the following Theorem 6 states that transitivity holds when passing through duals.

Theorem 6. *If $p \bowtie p'$ and $\mathrm{co}(p') \bowtie q$, then $p \bowtie q$.*

We now show that the dual is maximal w.r.t. the subtyping relation, like the dual in the untimed setting. We start by defining the semantic subyting preorder, which is a sound and complete model of the Gay and Hole subtyping relation (in reverse order) for untimed session types [4]. Intuitively, p is subtype of q if every q' compliant with q is compliant with p, too.

Definition 10 (Semantic subtyping). *For all TSTs p, we define the set $p^{\bowtie}$ as $\{q \mid p \bowtie q\}$. Then, we define the relation $p \sqsubseteq q$ whenever $p^{\bowtie} \supseteq q^{\bowtie}$.*

The following theorem states that $\mathrm{co}(p)$ is the maximum (i.e., the most "precise") in the set of the compliants of p, if not empty.

Theorem 7. $q \bowtie p \implies q \sqsubseteq \mathrm{co}(p)$

The following theorem reduces the problem of deciding $p \sqsubseteq q$ to that of checking compliance between p and $\mathrm{co}(q)$. Since both compliance and the dual construction are decidable, this implies decidability of subtyping.

Theorem 8. *If q admits a compliant, then: $p \sqsubseteq q \iff p \bowtie \mathrm{co}(q)$.*

4 Runtime monitoring

In this section we study runtime monitoring based on TSTs. The setting is the following: two participants A and B want to interact according to two (compliant) TSTs p_{A} and p_{B}, respectively. This interaction happens through a server, which monitors all the messages exchanged between A and B, while keeping track of the passing of time. If a participant (say, A) sends a message not expected by her TST, then the monitor classifies A as *culpable* of a violation. There are other two circumstances where A is culpable: (i) p_{A} is an internal choice, but A loses time until all the branches become unfeasible, or (ii) p_{A} is an external choice, but A does not readily receive an incoming message sent by B.

Note that the semantics in Figure 1 cannot be directly exploited to define such a runtime monitor, for two reasons. First, the synchronisation rule is purely symmetric, while the monitor outlined above assumes an asymmetry between internal and external choices. Second, the semantics in Figure 1 does not have transitions (either messages or delays) which are not allowed by the TSTs: for instance, $(\mathtt{!a}\{t \leq 1\}, \nu)$ cannot take any transitions (neither $\mathtt{!a}$ nor δ) if $\nu(t) > 1$. In a runtime monitor we want to avoid such kind of situations, where no actions are possible, and the time is frozen. More specifically, our desideratum is that the runtime monitor acts as a *deterministic* automaton, which reads a *timed trace* (a sequence of actions and time delays) and it reaches a unique state γ, which can be inspected to find which of the two participants (if any) is culpable.

To reach this goal, we define the semantics of the runtime monitor on two levels. The first level, specified by the relation $\twoheadrightarrow$, deals with the case of honest participants; however, differently from the semantics in Section 2, here we decouple the action of sending from that of receiving. More precisely, if A has an internal choice and B has an external choice, then we postulate that A must move first, by doing one of the outputs in her choice, and then B must be ready to do the corresponding input. The second level, called *monitoring semantics* and specified by the relation $\twoheadrightarrow_M$, builds upon the first one. Each move accepted by the first level is also accepted by the monitor. Additionally, the monitoring semantics defines transitions for actions not accepted by the first level, for instance unexpected input/output actions, and improper time delays. In these cases, the monitoring semantics signals which of the two participants is culpable.

Definition 11 (Monitoring semantics of TSTs). Monitoring configurations $\gamma, \gamma', \ldots$ *are terms of the form $P \parallel Q$, P and Q are triples (p, c, ν), where*

$$(\mathtt{!a}\{g,R\}.\,p \oplus p',[],\nu) \parallel (q,[],\eta) \xrightarrow{\mathsf{A}:\mathtt{!a}}\!\!\!\!\rightarrow (p,[\mathtt{!a}],\nu[R]) \parallel (q,[],\eta) \quad \text{if } \nu \in [\![g]\!] \qquad \text{[M-}\oplus\text{]}$$

$$(p,[\mathtt{!a}],\nu) \parallel (\mathtt{?a}\{g,R\}.\,q + q',[],\eta) \xrightarrow{\mathsf{B}:\mathtt{?a}}\!\!\!\!\rightarrow (p,[],\nu) \parallel (q,[],\eta[R]) \quad \text{if } \nu \in [\![g]\!] \qquad \text{[M-+]}$$

$$\frac{\nu+\delta \in \mathtt{rdy}(p) \qquad \eta+\delta \in \mathtt{rdy}(q)}{(p,[],\nu) \parallel (q,[],\eta) \xrightarrow{\delta}\!\!\!\!\rightarrow (p,[],\nu+\delta) \parallel (q,[],\eta+\delta)} \qquad \text{[M-Del]}$$

$$\frac{(p,c,\nu) \parallel (q,d,\eta) \xrightarrow{\lambda}\!\!\!\!\rightarrow (p',c',\nu') \parallel (q',d',\eta')}{(p,c,\nu) \parallel (q,d,\eta) \xrightarrow{\lambda}\!\!\!\!\rightarrow_M (p',c',\nu') \parallel (q',d',\eta')} \qquad \text{[M-Ok]}$$

$$\frac{(p,c,\nu) \parallel (q,d,\eta) \not\xrightarrow{\mathsf{A}:\ell}\!\!\!\!\rightarrow}{(p,c,\nu) \parallel (q,d,\eta) \xrightarrow{\mathsf{A}:\ell}\!\!\!\!\rightarrow_M (\mathbf{0},c,\nu) \parallel (q,d,\eta)} \qquad \text{[M-FailA]}$$

$$\frac{(d=[] \wedge \nu+\delta \notin \mathtt{rdy}(p)) \;\vee\; d \neq []}{(p,c,\nu) \parallel (q,d,\eta) \xrightarrow{\delta}\!\!\!\!\rightarrow_M (\mathbf{0},c,\nu+\delta) \parallel (q,d,\eta+\delta)} \qquad \text{[M-FailD]}$$

Fig. 4. Monitoring semantics (symmetric rules omitted).

p is either a TST or $\mathbf{0}$, and c is a one-position buffer (either empty or containing an output label). The transition relations $\twoheadrightarrow$ and $\twoheadrightarrow_M$ over monitoring configurations, with labels $\lambda, \lambda', \ldots \in (\{\mathsf{A},\mathsf{B}\} \times \mathtt{L}) \cup \mathbb{R}_{\geq 0}$, is defined in Figure 4.

In the rules in Figure 4, we always assume that the leftmost TST is governed by A, while the rightmost one is governed by B. In rule [M-⊕], A has an internal choice, and she can fire one of her outputs `!a`, provided that its buffer is empty, and the guard g is satisfied. When this happens, the message `!a` is written to the buffer, and the clocks in R are reset. Then, B can read the buffer, by firing `?a` in an external choice through rule [M-+]; this requires that the buffer of B is empty, and the guard g of the branch `?a` is satisfied. Rule [M-Del] allows time to pass, provided that the delay δ is permitted for both participants, and both buffers are empty. The last three rules specify the runtime monitor. Rule [M-Ok] says that any move accepted by $\twoheadrightarrow$ is also accepted by the monitor. Rule [M-FailA] is used when participant A attempts to do an action not permitted by $\twoheadrightarrow$: this makes the monitor evolve to a configuration where A is culpable (denoted by the term $\mathbf{0}$). Rule [M-FailD] makes A culpable when time passes, in two cases: either A has an internal choice, but the guards are no longer satisfiable; or she has an external choice, and there is an incoming message.

When both participants behave honestly, i.e., they never take [M-Fail*] moves, the monitoring semantics preserves compliance (Theorem 9). The *monitoring compliance* relation $\bowtie_M$ is the straightforward adaptation of that in Definition 6, except that $\twoheadrightarrow$ transitions are used instead of $\rightarrow$ ones (see [5]).

Theorem 9. $\bowtie \;= \bowtie_M$.

The following lemma establishes that the monitoring semantics is deterministic: that is, if $\gamma \xrightarrow{\lambda}\!\!\!\!\rightarrow_M \gamma'$ and $\gamma \xrightarrow{\lambda}\!\!\!\!\rightarrow_M \gamma''$, then $\gamma' = \gamma''$. Determinism is a very desirable property indeed, because it ensures that the culpability of a participant at any given time is uniquely determined by the past actions. Furthermore, for all finite timed traces $\boldsymbol{\lambda}$ (i.e., sequences of actions $\mathsf{A} : \ell$ or time delays δ), there exists some configuration γ reachable from the initial one.

Lemma 2. *Let $\gamma_0 = (p, [], \nu_0) \parallel (q, [], \eta_0)$. If $p \bowtie q$, then $(\twoheadrightarrow_M, \gamma_0)$ is deterministic, and for all finite timed traces $\boldsymbol{\lambda}$ there exists (unique) γ such that $\gamma_0 \xrightarrow{\boldsymbol{\lambda}}\!\!\!\!\rightarrow_M \gamma$.*

The goal of the runtime monitor is to detect, at any state of the execution, which of the two participants is culpable (if any). Further, we want to identify who is in charge of the next move. This is formalised by the following definition.

Definition 12 (Duties & culpability). *Let $\gamma = (p, c, \nu) \parallel (q, d, \eta)$. We say that A is* culpable *in γ iff $p = \mathbf{0}$. We say that A is* on duty *in γ if (i) A is not culpable in γ, and (ii) either p is an internal choice, or d is not empty.*

Lemma 3 guarantees that, in each reachable configuration, only one of the participants can be on duty; and if no one is on duty nor culpable, then both participants have reached success.

Lemma 3. *If $p \bowtie q$ and $(p, [], \nu_0) \parallel (q, [], \eta_0) \twoheadrightarrow_M^* \gamma$, then:*

1. *there exists at most one participant on duty in γ,*
2. *if there exists some culpable participants in γ, then no one is on duty in γ,*
3. *if no one is on duty in γ, then γ is success, or someone is culpable in γ.*

Note that both participants may be culpable in a configuration. E.g., let $\gamma = (\mathtt{!a}\{\mathtt{true}\}, [], \eta_0) \parallel (\mathtt{?a}\{\mathtt{true}\}, [], \eta_0)$. By applying [M-FailA] twice, we obtain:

$$\gamma \xrightarrow{\mathsf{A}:\mathtt{?b}}\!\!\!\!\rightarrow_M (\mathbf{0}, [], \nu_0) \parallel (\mathtt{?a}\{\mathtt{true}\}, [], \eta_0) \xrightarrow{\mathsf{B}:\mathtt{?b}}\!\!\!\!\rightarrow_M (\mathbf{0}, [], \nu_0) \parallel (\mathbf{0}, [], \eta_0)$$

and in the final configuration both participants are culpable.

Example 9. Let $p = \mathtt{!a}\{2 < t < 4\}$ be the TST of participant A, and let $q = \mathtt{?a}\{2 < t < 5\} + \mathtt{?b}\{2 < t < 5\}$ be that of B. We have that $p \bowtie q$. Let $\gamma_0 = (p, [], \nu_0) \parallel (q, [], \nu_0)$. A correct interaction is given by the timed trace $\eta = \langle 1.2,\ \mathsf{A} : \mathtt{!a},\ \mathsf{B} : \mathtt{?a} \rangle$. Indeed, $\gamma_0 \xrightarrow{\eta}\!\!\!\!\rightarrow_M (\mathbf{1}, [], \nu_0) \parallel (\mathbf{1}, [], \nu_0)$. On the contrary, things may go awry in three cases:

(i) a participant does something not permitted. E.g., if A fires $\mathtt{a}$ at 1 t.u., by [M-FailA]: $\gamma_0 \xrightarrow{1}\!\!\!\!\rightarrow_M \xrightarrow{\mathsf{A}:\mathtt{!a}}\!\!\!\!\rightarrow_M (\mathbf{0}, [], \nu_0 + 1) \parallel (q, [], \eta_0 + 1)$, where A is culpable.

(ii) a participant avoids to do something she is supposed to do. E.g., assume that after 6 t.u., A has not yet fired $\mathtt{a}$. By rule [M-FailD], we obtain $\gamma_0 \xrightarrow{6}\!\!\!\!\rightarrow_M (\mathbf{0}, [], \nu_0 + 6) \parallel (q, [], \eta_0 + 6)$, where A is culpable.

(iii) a participant does not receive a message as soon as it is sent. For instance, after $\mathtt{a}$ is sent at 1.2 t.u., at 5.2 t.u. B has not yet fired $\mathtt{?a}$. By [M-FailD], $\gamma_0 \xrightarrow{1.2}\!\!\!\!\rightarrow_M \xrightarrow{\mathsf{A}:\mathtt{!a}}\!\!\!\!\rightarrow_M \xrightarrow{4}\!\!\!\!\rightarrow_M (\mathbf{1}, [\mathtt{!a}], \nu_0 + 5.2) \parallel (\mathbf{0}, [], \eta_0 + 5.2)$, where B is culpable.

5 Conclusions

We have studied a theory of session types (TSTs), featuring timed synchronous communication between two endpoints. We have defined a decidable notion of compliance between TSTs, a decidable procedure to detect when a TST admits a compliant, a decidable subtyping relation, and a (decidable) runtime monitoring.

All these notions have been exploited in the design and development of a message-oriented middleware which uses TSTs to drive safe interactions among distributed components. The idea is a contract-oriented, bottom-up composition, where only those services with compliant contracts can interact via (binary) sessions. The middleware makes available a global store where services can advertise contracts, in the form of TSTs. Assume that A advertises a contract p to the store (this is only possible if p admits a compliant). A session between A and B can be established if (i) B advertises a contract q compliant with p, or (ii) B *accepts* the contract p (in this case, the contract of B is the dual of p). When the session is established, A and B can interact by sending/receiving messages through the session. During the interaction, all their actions are monitored (according to Definition 11), and possible misbehaviours are detected (according to Definition 12). The middleware is accessible through a set of public APIs; a suite of tools for developing contract-oriented applications is available at [5].

Related work. Compliance between TSTs is loosely related to the notion of compliance between *untimed* session types (in symbols, $\bowtie_u$). Let $u(p)$ be the session type obtained by erasing from p all the timing annotations. It is easy to check that the semantics of $(u(p), \nu_0) \mid (u(q), \nu_0)$ in Section 2 coincides with the semantics of $u(p) \mid u(q)$ in [4]. Therefore, if $u(p) \bowtie u(q)$, then $u(p) \bowtie_u u(q)$. Instead, *semantic conservation* of compliance does not hold, i.e. it is not true in general that if $p \bowtie q$, then $u(p) \bowtie_u u(q)$. E.g., let $p = \texttt{!a}\{t < 5\} \oplus \texttt{!b}\{t < 0\}$, and let $q = \texttt{?a}\{t < 7\}$. We have that $p \bowtie q$ (because the branch !b can never be chosen), whereas $u(p) = \texttt{!a} \oplus \texttt{!b} \not\bowtie_u \texttt{?a} = u(q)$. Note that, for every p, $u(\mathrm{co}(p)) = \mathrm{co}(u(p))$.

In the context of session types, time has been originally introduced in [9]. However, the setting is different than ours (multiparty and asynchronous, while ours is bi-party and synchronous), as well as its objectives: while we have focussed on primitives for the bottom-up approach to service composition [6], [9] extends to the timed case the *top-down* approach. There, a *choreography* (expressing the overall communication behaviour of a set of participants) is projected into a set of *session types*, which in turn are refined as processes, to be type-checked against their session type in order to make service composition preserve the properties enjoyed by the choreography.

Our approach is a conservative extension of untimed session types, in the sense that a participant which performs an output action chooses not only the branch, but the time of writing too; dually, when performing an input, one has to passively follow the choice of the other participant. Instead, in [9] external choices can also delay the reading time. The notion of correct interaction studied in [9] is called *feasibility*: a choreography is feasible iff all its reducts can reach the

success state. This property implies progress, but it is undecidable in general, as shown by [20] in the context of communicating timed automata (however, feasibility is decidable for the subclass of *infinitely satisfiable* choreographies). The problem of deciding if, given a local type T, there exists a choreography G such that T is in the projection of G and G enjoys (global) progress is not being addressed in [9]. We think that it can be solved by adapting our kind system (in particular rule [T-+] must be adjusted).

Another problem not addresses by [9] is that of determinining if a set of session types enjoys progress (which, as feasibility of choreographies, would be undecidable). In our work we have considered this problem, under a synchronous semantics, and with the restriction of two participants. Extending our semantics to an asynchronous one would make compliance undecidable (as it is for untimed asynchronous session types [15]). Note that our progress-based notion of compliance does not imply progress with the semantics of [9] (adapted to the binary case). For instance, let $p = \mathtt{?a}\{x \leq 2\}.\,\mathtt{!a}\{x \leq 1\}$ and $q = \mathtt{!a}\{y \leq 1\}.\,\mathtt{?a}\{y \leq 1\}$. We have that $p \bowtie q$, while in the semantics of [9] $(\nu_0, (p, q, \boldsymbol{w_0})) \rightarrow^* (\nu, (\mathtt{!a}\{x \leq 1\}, \mathtt{?a}\{y \leq 1\}, \boldsymbol{w_0}))$ with $\nu(x) = \nu(y) > 1$, which is a deadlock state.

Dynamic verification of timed multiparty session types is addressed by [22], where the top-down approach to service composition is pursued [19]. Our middleware instead composes and monitors services in a bottom-up fashion [6].

In [13] timed specifications are studied in the setting of *timed I/O transition systems* (TIOTS). They feature a notion of correct composition, called *compatibility*, following the *optimistic approach* pursued in [14]: roughly, two systems are compatible whenever there exists an environment which, composed with them, makes "undesirable" states unreachable. A notion of *refinement* is coinductively formalised as an alternating timed simulation. Refinement is a preorder, and it is included in the semantic subtyping relation (using compatibility instead of $\bowtie$). Because of the different assumptions (open systems and broadcast communications in [13], closed binary systems in TSTs), compatibility/refinement seem unrelated to our notions of compliance/subtyping. Despite the main notions in [13] are defined on semantic objects (TIOTS), they can be decided on timed I/O automata, which are finite representations of TIOTS. With respect to TSTs, timed I/O automata are more liberal: e.g., they allow for *mixed choices*, while in TSTs each state is either an input or an output. However, this increased expressiveness does not seem appropriate for our purposes: first, it makes the concept of culpability unclear (and it breaks one of the main properties of ours, i.e. that at most one participant is on duty at each execution step); second, it seems to invalidate any dual construction. This is particularly unwelcome, since this construction is one of the crucial primitives of contract-oriented interactions.

Acknowledgments. Work partially supported by Aut. Reg. of Sardinia L.R.7/2007 CRP-17285 (TRICS), P.I.A. 2010 ("Social Glue"), P.O.R. Sardegna F.S.E. Operational Programme of the Aut. Reg. of Sardinia, EU Social Fund 2007-13 – Axis IV Human Resources, Objective l.3, Line of Activity l.3.1), by MIUR PRIN 2010-11 project "Security Horizons", and by EU COST Action IC1201 "Behavioural Types for Reliable Large-Scale Software Systems" (BETTY).

References

1. PayPal buyer protection. `https://www.paypal.com/us/webapps/mpp/ua/useragreement-full#13`. Accessed: January 20, 2015.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
3. F. Barbanera and U. de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In *PPDP*, pages 155–164, 2010.
4. F. Barbanera and U. de'Liguoro. Sub-behaviour relations for session-based client/server systems. *Math. Struct. in Comp. Science*, pages 1–43, 1 2015.
5. M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. Compliance and subtyping in timed session types. Technical report, 2015. `co2.unica.it`.
6. M. Bartoletti, E. Tuosto, and R. Zunino. Contract-oriented computing in CO_2. *Sci. Ann. Comp. Sci.*, 22(1), 2012.
7. G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *Formal methods for the design of real-time systems*, pages 200–236. Springer, 2004.
8. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *ACPN*, pages 87–124, 2003.
9. L. Bocchi, W. Yang, and N. Yoshida. Timed multiparty session types. In *CONCUR*, pages 419–434, 2014.
10. G. Castagna, M. Dezani-Ciancaglini, E. Giachino, and L. Padovani. Foundations of session types. In *PPDP*, pages 219–230, 2009.
11. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. *ACM Transactions on Programming Languages and Systems*, 31(5), 2009.
12. R. Corin, P.-M. Deniélou, C. Fournet, K. Bhargavan, and J. J. Leifer. A secure compiler for session abstractions. *Journal of Computer Security*, 16(5), 2008.
13. A. David, K. G. Larsen, A. Legay, U. Nyman, L. Traonouez, and A. Wasowski. Real-time specifications. *STTT*, 17(1):17–45, 2015.
14. L. de Alfaro and T. A. Henzinger. Interface automata. In *ACM SIGSOFT*, pages 109–120, 2001.
15. P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP*, 2013.
16. M. Dezani-Ciancaglini and U. de'Liguoro. Sessions and session types: An overview. In *WS-FM*, pages 1–28, 2009.
17. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994.
18. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP*, 1998.
19. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, 2008.
20. P. Krcál and W. Yi. Communicating timed automata: The more synchronous, the more difficult to verify. In *CAV*, pages 249–262, 2006.
21. C. Laneve and L. Padovani. The *must* preorder revisited. In *CONCUR*, pages 212–225, 2007.
22. R. Neykova, L. Bocchi, and N. Yoshida. Timed runtime monitoring for multiparty conversations. In *BEAT*, pages 19–26, 2014.
23. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE*, 1994.
24. N. Yoshida, R. Hu, R. Neykova, and N. Ng. The Scribble protocol language. In *TGC*, pages 22–41, 2013.