Compliance in Behavioural Contracts

a brief survey

Massimo Bartoletti[1], Tiziana Cimoli[1], and Roberto Zunino[2]

[1] Università degli Studi di Cagliari, Italy
[2] Università degli Studi di Trento, Italy

Abstract. Behavioural contracts are formal specifications of interaction protocols between two or more distributed services. Despite the heterogeneous nature of the formalisms for behavioural contracts that have appeared in the literature, most of them feature a notion of *compliance*, which characterises when two or more contracts lead to correct interactions between services respecting them. We discuss and compare a selection of these notions in four different models of contracts: τ-less CCS, session types, interface automata, and contract automata.

1 Introduction

Several recent works study *behavioural contracts* as a tool to formalise and discipline correct interactions between distributed services [46]. Many of these works define, or build upon, some notion of *compliance* (also called *duality*, *conformance*, or *agreement*) between two or more contracts. Intuitively, compliance between contracts guarantees that services respecting them will interact "correctly", according to some notion of correctness which varies from approach to approach. This notion is exploited e.g., to type-check whether the specification of a service respects its contracts [43,44,42,45,33], or to dynamically compose services with compliant ones [18,16,15].

To choose the most suitable notion of "correct behaviour" for a given distributed application, it would be desirable to have a clear understanding of the actual properties enjoyed by various notions of compliance, and of the relations among them. This is not an easy task, because the ecosystem of notions proposed in the literature is wide and heterogeneous. Indeed, many different compliance relations have been considered in the literature, and they have been defined on, or applied to, a variety of different languages and formalisms, among which session-types [23,22,17], Petri nets [56,8], process algebras [26,27,47,28,34] and various automata-based models [37,21,20,51], among others.

In this paper we start a systematic investigation of compliance relations between behavioural contracts. We aim for a semantic, language-independent analysis, which abstracts from the actual formalism wherein contracts are given meaning. Along the lines of the treatment of behavioural equivalences and preorders in concurrency theory, we model contracts as states in Labelled Transition Systems (LTSs), with labels ranging over input, output, and internal actions.

This interpretation is straightforward for some models of contracts (e.g., interface automata [37] are just finite-state LTSs), while others can be dealt with by relating them to LTSs (e.g., session types and τ-less CCS processes induce an LTS through their operational semantics). By exploiting this common ground, we formalise different notions of compliance as relations between LTS states, and we compare them in four classes of contracts: τ-less CCS [39], session types [43], interface automata [37], and contract automata [20]. The results of our investigation are reported in Section 4, and an overview of other related approaches, not yet included in our formal comparison, is given in Section 5.

2 Contracts

In this section we provide a unifying ground for behavioural contracts. They will be formalised as states of a Labelled Transition System (LTS) where labels are partitioned into *internal*, *input*, and *output* actions. We will show that contracts expressed in other formalisms (e.g., τ-less CCS and session types) can be interpreted as states of this LTS. All the compliance relations defined later on in Section 3. will be formalised as binary relations between states.

2.1 Basics

Our treatment is developed within the LTS $\big(\mathbb{U}, \mathsf{A}_\tau, \{\xrightarrow{\ell_\tau} \mid \ell_\tau \in \mathsf{A}_\tau\}\big)$, where:

- $\mathbb{U}$ is the universe of *states* (ranged over by $p, q, \ldots$), also called *contracts*;
- A_τ (ranged over by $\ell_\tau, \ell'_\tau, \ldots$) is the set of *labels*, partitioned into *input actions* $?\mathsf{a}, ?\mathsf{b}, \ldots \in \mathsf{A}^?$, *output actions* $!\mathsf{a}, !\mathsf{b}, \ldots \in \mathsf{A}^!$, and the *internal action* τ;
- $\xrightarrow{\ell_\tau} \subseteq \mathbb{U} \times \mathbb{U}$ is a *transition relation*, for all ℓ_τ.

We let $\ell, \ell', \ldots$ range over $\mathsf{A} = \mathsf{A}^? \cup \mathsf{A}^!$. We postulate an involution $\mathrm{co}(\cdot)$ on A, such that $\mathrm{co}(?\mathsf{a}) = !\mathsf{a}$ and $\mathrm{co}(!\mathsf{a}) = ?\mathsf{a}$. The *reducts of* p are the states reachable from p with an arbitrary sequence of transitions; we say that p is *finite-state* when the set of its reducts is finite. A *trace* is a (possibly infinite) sequence $p_0 \xrightarrow{{\ell_\tau}^{(1)}} p_1 \xrightarrow{{\ell_\tau}^{(2)}} \cdots$. A τ-trace is a trace where ${\ell_\tau}^{(i)} = \tau$, for all i (similarly for τ-reduct). We denote with $\mathbf{0}$ a state with no outgoing transitions, and we will interpret $\mathbf{0}$ as the *success* state. We show some (finite-state) contracts in Figure 1.

Notation 1 *We adopt the following notation:*

- $\mathcal{R}^*$ *for the reflexive and transitive closure of a relation* $\mathcal{R}$
- $p \xrightarrow{\ell_\tau}$ *when* $\exists p' \,.\, p \xrightarrow{\ell_\tau} p'$. *Further, we write* $p \not\to$ *when* $\exists \ell_\tau \,.\, p \xrightarrow{\ell_\tau}$
- *for a set* $L \subseteq \mathsf{A}$, *we define* $L^? = L \cap \mathsf{A}^?$ *and* $L^! = L \cap \mathsf{A}^!$
- $\Rightarrow = (\xrightarrow{\tau})^*$ *is the* weak transition relation. *We define* $\overset{\ell_\tau}{\Rightarrow}$ *as* $\Rightarrow\xrightarrow{\ell_\tau}\Rightarrow$
- $p{\downarrow} = \{\ell \mid p \xrightarrow{\ell}\}$ *are the* barbs of p, *and* $p{\Downarrow} = \{\ell \mid p \overset{\ell}{\Rightarrow}\}$ *are its* weak barbs
- $p{\uparrow}$ *is true when* p *has an infinite internal computation* $p \xrightarrow{\tau} p_1 \xrightarrow{\tau} p_2 \xrightarrow{\tau} \cdots$

The above notation for $\to$ *is extended to* $\Rightarrow$ *as expected.*

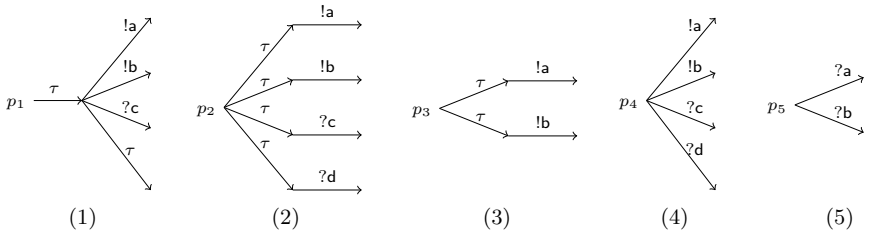


Fig. 1: Some contracts.

Two contracts can be composed with the operator $\|$, which formalises the standard synchronisation à la CCS [52].

Definition 1 (Parallel composition). *For all $p, q \in \mathbb{U}$, we define the* parallel composition $p \parallel q$ *as the state in $\mathbb{U}$ whose transitions are given by the rules:*

$$\frac{p \xrightarrow{\ell_\tau} p'}{p \parallel q \xrightarrow{\ell_\tau} p' \parallel q} \qquad \frac{q \xrightarrow{\ell_\tau} q'}{p \parallel q \xrightarrow{\ell_\tau} p \parallel q'} \qquad \frac{p \xrightarrow{\ell} p' \quad q \xrightarrow{\mathrm{co}(\ell)} q'}{p \parallel q \xrightarrow{\tau} p' \parallel q'}$$

We describe in the following subsections some classes of contracts which have been considered in the literature.

2.2 Interface Automata

As a first subclass of the set $\mathbb{U}$, we consider *interface automata* [37]. These are finite-state automata, which can communicate through the synchronization of input and output actions, and they can perform internal actions (possibly of different kinds). Synchronization of input and output actions is obtained in [37] by constructing the cartesian product of interface automata, with the restriction that matching input and output actions must fire simultaneously.

To adapt interface automata to our framework, we collapse the different internal actions to τ. Once this is done, interface automata simply correspond to finite-state contracts. Note that in our framework we do not need to explicitly construct the cartesian product, since we obtain a contract with the same behaviour through Definition 1. We then denote with IA the set of finite-state contracts (so, $\mathrm{IA} \subset \mathbb{U}$). For instance, all the contracts in Figure 1 belong to IA.

2.3 τ-less CCS

The contracts used in [32,47,34] are terms of the process calculus CCS without τ's [39]. Differently from Milner's CCS [52], these have two kinds of choice. In an *internal choice* $C \oplus D$ the process decides which one of the two branches to follow, whereas in an *external choice* $C \,\&\, D$, the decision is taken by the environment.

To have a simple embedding of τ-less CCS into our framework, we restrict our study to the fragment where choices are prefix-guarded.

Definition 2 (τ-less CCS). τ-less CCS processes *are terms with the syntax:*

$$C \;::=\; \&_{i\in I}\,\ell_i.C_i \;\mid\; \bigoplus_{i\in I}\ell_i.C_i \;\mid\; \mathrm{rec}_X\,C \;\mid\; X$$

where (i) the set I is finite, (ii) the actions ℓ_i in internal/external choices are pairwise distinct, and (iii) recursion is prefix-guarded.

We write $\mathbf{0}$ for the empty (internal/external) choice, and omit trailing occurrences of $\mathbf{0}$. We adopt the equi-recursive approach, by considering terms up-to unfolding of recursion. The semantics of τ-less CCS is given below.

Definition 3. *We denote with $\tau\mathrm{C}$ the set of contracts of the form C or $[\mathsf{!a}]\,C$, with C closed, and transitions given by the following rules:*

$$\&_{i\in I}\,\ell_i.C_i \xrightarrow{\ell_k} C_k \;\;(k\in I) \qquad \bigoplus_{i\in I}\ell_i.C_i \xrightarrow{\tau} [\ell_k]\,C_k \;\;(k\in I) \qquad [\ell]\,C \xrightarrow{\ell} C$$

An external choice can always perform each of its actions. An internal choice $\bigoplus_{i\in I}\ell_i.C_i$ must first commit to one of the branches $\ell_k.C_k$, and this produces a *committed choice* $[\ell_k]\,C_k$, which can only perform the action ℓ_k. As a consequence, a contract in $\tau\mathrm{C}$ may have several outgoing transitions (either input or output), but internal transitions cannot be mixed with input/output ones. There cannot be two internal transitions in a row, and after an internal transition, the target state will have exactly one outgoing transition. Contracts in $\tau\mathrm{C}$ are finite-state, so $\tau\mathrm{C} \subseteq \mathrm{IA}$.

Example 1. The process $C = \mathsf{!a} \oplus \mathsf{!b} \oplus \mathsf{?c} \oplus \mathsf{?d}$ is denoted by the contract p_2 in Figure 1, while the contract p_4 denotes the process $\mathsf{!a} \,\&\, \mathsf{!b} \,\&\, \mathsf{?c} \,\&\, \mathsf{?d}$: indeed, since the latter is an external choice, all the labels are enabled at the same time. Instead, the contract p_1 in Figure 1 does not belong to $\tau\mathrm{C}$, for two different reasons. First, it has an internal transition at the same level of a non-internal one; second, there are two consecutive internal transitions.

Note that, if we allowed internal actions in external choices, we would essentially turn the choice into an internal one. Indeed, the abstraction operator of [31] makes internal choices emerge from external ones. However, such abstraction produces contracts which go beyond $\tau\mathrm{C}$, and are instead contained in IA.

2.4 Session Types

Session types [43,44] are terms of a process algebra featuring a *selection* construct (i.e., an internal choice among a set of branches, each one performing some output), and a *branching* construct (i.e., an external choice among a set of inputs offered to the environment). With the restriction given in Section 2.3 (i.e., choices are prefix-guarded), and further assuming no channel passing, session types are just a special case of τ-less CCS contracts, where the actions in internal choices are all outputs, and the actions in external choices are all inputs.

Definition 4 (Session types). Session types *are τ-less CCS processes respecting the following syntax:*

$$T \quad ::= \quad \&_{i\in I} ?\mathsf{a}_i.T_i \;\mid\; \bigoplus_{i\in I} !\mathsf{a}_i.T_i \;\mid\; \mathrm{rec}_X\, T \;\mid\; X$$

We denote with ST the set of contracts of the form T or $[!\mathsf{a}]\,T$, with T closed and transitions given as in Definition 3. Note that all the outgoing transitions from a state must have the same (internal/input/output) kind. It is easy to check that ST $\subset \tau$C.

Example 2. The contract p_3 in Figure 1 represents the session type $!\mathsf{a} \oplus !\mathsf{b}$. Since it is an internal choice, according with Definition 3, there is a commit on the chosen branch before actually firing the output action. The contract p_4 does not belong to ST because it has two output transitions on the same level, and they are also mixed with input ones. The contract p_2 does not belong to ST, because it has a internal transition before an input transition.

While ST is a strict subset of τC, note that it is possible to encode the latter in the former, as shown in [48].

2.5 Contract Automata

Contract automata [20,19] are similar to interface automata, from which they differ in the interpretation of labels. Labels are either *requests*, modelling resources/interactions expected from the environment, or *offers*, modelling resources/interactions produced in exchange. Internal actions are not allowed. The interaction between contract automata is obtained through a composition operation, similar to that of interface automata. An interaction is considered *successful* if all the requested actions are met, while the offered actions may be ignored. Offered actions are considered to be available to the counterpart, which can either choose to use or ignore them. This is similar to the intuition of external choices in session types, so we model offers as actions in $\mathsf{A}^?$; symmetrically, we model requests as actions in $\mathsf{A}^!$. The contracts obtained from contract automata are finite-state and they have no internal actions. We denote them with CA.

Lemma 1. CA $\subset \tau$C, *and* ST $\cap$ CA $\neq \emptyset$.

Proof. Since a contract automaton is finite-state, one can regard it as a set of recursive equations of the form $X_i = \&_j \ell_j.C_j$, where C_j are τ-less CCS processes. These can be turned into a τC contract applying Bekič's lemma. To show that ST $\cap$ CA $\neq \emptyset$, it suffices to pick p_5 in Figure 1. □

Example 3. The contract p_4 in Figure 1 belongs to CA: it fires input and output actions at the same level and no τ is present. On the contrary, the contracts p_1, p_2 and p_3 do not belong to CA, because of the presence of internal transactions.

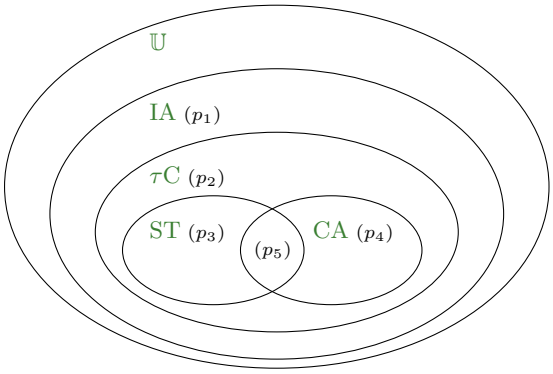


Fig. 2: Relations between some classes of contracts (all the inclusions are strict).

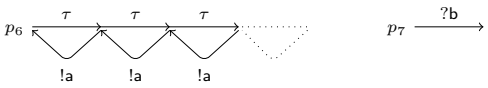
2.6 Relations between classes of contracts

Figure 2 relates the classes of contracts considered in the previous subsections, and the derived class $\tau\mathrm{C}^{-} = \tau\mathrm{C} \setminus (\mathrm{ST} \cup \mathrm{CA})$. To show that all the inclusions are strict, consider the contracts in Figure 1. We have that $p_1 \in \mathrm{IA} \setminus \tau\mathrm{C}$; $p_2 \in \tau\mathrm{C} \setminus \{\mathrm{ST} \cup \mathrm{CA}\}$; $p_3 \in \mathrm{ST} \setminus \mathrm{CA}$; $p_4 \in \mathrm{CA} \setminus \mathrm{ST}$; $p_5 \in \mathrm{ST} \cap \mathrm{CA}$. To show $\mathrm{IA} \subset \mathbb{U}$, it suffices to take an infinite-state contract.

3 Compliance relations

In this section we survey some notions of compliance that have appeared in the literature, and we formalise them as relations between contracts in the LTS of Section 2. Since many of the original definitions apply to specific models, in order to unify the study of these notions we had to adapt some of them. For instance, some works focus on symmetric relations, while some others on *asymmetric* ones, where one of the two contracts (e.g., playing the role of a client), may be allowed to terminate interaction or to skip messages despite of the state of the other one (e.g., playing the role of a server). For uniformity, in this paper we only consider asymmetric relations. In most cases, a symmetric compliance relation $\bowtie$ can be obtained by intersecting asymmetric compliance $\lhd$ with its inverse relation (a notable exception is $\lhd_{may}$ in Definition 6). Further, although some notions of compliance in the literature are naturally multi-party (e.g., compatibility in IA [37]), for uniformity we restrict them to the *binary* case, where only two participants are involved.

Progress. We start by considering the notion of *progress*, where compliance is interpreted as the absence of deadlock (on the client-side, since we are considering the asymmetric relation). Formally, in Definition 5 we say that a contract p has progress with q (in symbols, $p \lhd_{pg} q$) iff, whenever a τ-reduct of $p \parallel q$ cannot take internal transitions, then p has reached the success state.



(6) (7)

Fig. 3: Two session types with an asynchronous semantics.

Definition 5 (Progress). *We write* $p \lhd_{pg} q$ *iff:*

$$p \parallel q \;\Rightarrow\; p' \parallel q' \not\xrightarrow{\tau} \quad \textit{implies} \quad p' = \mathbf{0}$$

Example 4. Consider the pair of contracts (10) in Figure 6. When composed together, p_{10} and q_{10} can synchronise on label a, after which they both reach success. Hence, $p_{10} \bowtie_{pg} q_{10}$. The two contracts in (12) can synchronise forever, so $p_{12} \bowtie_{pg} q_{12}$. Consider now the pair (13): when composed together, we have $p_{13} \parallel q_{13} \xrightarrow{\tau} \xrightarrow{\tau} p' \parallel \mathbf{0} \not\xrightarrow{\tau}$, with $p' \neq \mathbf{0}$. Therefore, $p_{13} \not\lhd_{pg} q_{13}$, while $q_{13} \lhd_{pg} p_{13}$.

This notion has been used e.g. in τ-less CCS [34], in session types (both untimed [3] and timed [7]), and in types for CaSPiS [1]. Note that in *asynchronous* session types [36] the progress-based compliance of Definition 5 can relate contracts which arguably admit correct interactions. For instance, consider the session types $C_6 = \mathrm{rec}_X\, !\mathsf{a}.X$ and $C_7 = ?\mathsf{b}$, and let p_6 and p_7 (displayed in Figure 3) denote their asynchronous semantics. In p_6, internal actions are used to enqueue outputs in a FIFO buffer; queued outputs can then be fired. The asynchronous semantics of C_7 (denoted by p_7) is identical to the synchronous one. Clearly, $C_6 \not\bowtie_{pg} C_7$, because $C_6 \parallel C_7$ is stuck. Instead, we have that $p_6 \bowtie_{pg} p_7$, because the interaction of the two contracts produces an infinite τ-trace, even if no synchronisation ever happens. Stricter notions of compliance, like e.g. those in Definitions 8 to 10, manage to avoid such "vacuous" progress, where two contracts merely advance via internal τ-transitions (without ever synchronising).

May-testing compliance. The following three notions of compliance (Definitions 6 to 8) are inspired by the theory of testing in concurrent systems [38]. In Definition 6, a contract p is said *may-testing compliant* with q (in symbols, $p \lhd_{may} q$) if there exists a finite τ-trace of $p \parallel q$ which leads p to the success state.

Definition 6 (May-testing compliance). *We write* $p \lhd_{may} q$ *iff*

$$\exists q' \;.\; p \parallel q \;\Rightarrow\; \mathbf{0} \parallel q'$$

Example 5. The pair of contracts (15) in Figure 6 can synchronise on a and then succeed (on both sides); hence, $p_{15} \bowtie_{may} q_{15}$. In (12), both contracts can fire actions in a loop but they cannot terminate, so $p_{12} \not\lhd_{may} q_{12}$ and $q_{12} \not\lhd_{may} p_{12}$.

May-testing compliance assumes a *cooperative* scenario, where all the participants collaborate to achieve a common goal: if there is a way for the interaction

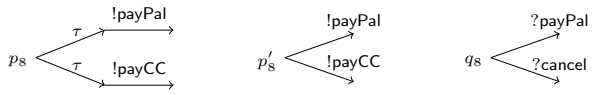


Fig. 4: Two client contracts p_8, p'_8 and one service contract q_8.

to succeed, it is enough, disregarding all the possible unsuccessful interactions. This requires participants to pre-agree on their internal choices, and the scheduler to only permit the synchronisations leading to success. For instance, in (15) the scheduler must not allow p_{15} and q_{15} to synchronise on b, since this would prevent p_{15} from reaching the success state. The $\lhd_{may}$ relation is similar to the *agreement* relation used in CA [19], under the assumption that there are only two contracts, and there exists a unique success state.

Example 6. Figure 4 shows two different ways to represent choices. The contracts p_8 and p'_8 model two clients of an online store, which want to pay by PayPal (payPal) or by credit card (payCC). Instead, the contract q_8 models an online store which accepts payments with PayPal, or it allows clients to cancel the transaction. Using progress-based compliance, we have $p_8 \not\lhd_{pg} q_8$ and $p'_8 \lhd_{pg} q_8$. Instead, using may-testing compliance, we have that $p_8 \lhd_{may} q_8$ and $p'_8 \lhd_{may} q_8$. From this we may observe two facts. First, under progress-based compliance, p_8 and p'_8 represent two different kinds of choice. The τ actions in p_8 force Definition 5 to consider *both* paths; instead, in p'_8 the transition $!\mathsf{payCC}$ can be ignored when testing compliance with q_8, because no synchronisation on payCC can happen. So, in a sense the choice in p_8 is made by the client, while that in p'_8 is made by the scheduler (or by the server). Second, note that may-testing compliance cannot discriminate between the two kinds of choice: indeed, Definition 6 assumes that all the choices are performed by an oracle, which always follows the path (if any) leading to success.

Must-testing compliance. The notion of compliance in [2] is inspired to must-testing [38]. This relation is stricter than may-testing, because it requires a contract to reach success in *all* (sufficiently long) traces. Formally, we say that a τ-trace $r_0 \to r_1 \to \cdots$ is *maximal* if it is infinite, or if it ends in a state r_n such that $r_n \overset{\tau}{\not\to}$. A contract p is must-testing compliant with q (in symbols, $p \lhd_{mst} q$) if, in all the maximal τ-traces of $p \parallel q$, the contract p reaches the success state.

Definition 7 (Must-testing compliance). *We write* $p_0 \lhd_{mst} q_0$ *iff*

$$\text{for all maximal } \tau\text{-traces } p_0 \parallel q_0 \xrightarrow{\tau} p_1 \parallel q_1 \xrightarrow{\tau} \cdots \quad : \quad \exists i \geq 0 \,.\, p_i = \mathbf{0}$$

Example 7. Consider the pair of contracts (16) in Figure 6. We have $p_{16} \bowtie_{mst} q_{16}$: indeed, there are two finite maximal τ-traces, and they lead both contracts to success. For the pair (20), there are two infinite maximal τ-traces. In the first trace, there is a finite number of synchronisations on a, then one synchronisation

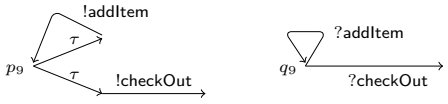


Fig. 5: The contract of an online store (q_9) and of a client (p_9).

on b, and finally the contract q_{20} loops forever with internal moves. Here, the contract p_{20} reaches success after the synchronisation on b. In the other maximal τ-trace, the two contracts synchronise on a forever, but p_{20} never reaches success. Hence, we have that $p_{20} \not\lhd_{mst} q_{20}$ and $q_{20} \not\lhd_{mst} p_{20}$.

Compared to may-testing compliance, in must-testing compliance we are no longer assuming that participants are cooperative: indeed, p must succeed, whatever internal choices q takes, and whatever synchronisations the scheduler enables. In a sense, all the choices (of q, of the scheduler, and also of p) are considered *demonic*, while in may-testing compliance they are all *angelic*.

Example 8. Consider the contract q_9 of an online store, and the contract p_9 of one of its clients. The client can iteratively add items to the shopping cart, or eventually choose to check out (and succeed). On the other side, the online store acknowledges the chosen items, and it succeeds when the client checks out. These contracts can be described in ST as $p_9 = \mathrm{rec}_X\, !\mathsf{addItem}.X \oplus !\mathsf{checkOut}$, and $q_9 = \mathrm{rec}_X\, ?\mathsf{addItem}.X \,\&\, ?\mathsf{checkOut}$. Note that there exists a maximal (infinite) trace where the client always chooses to add a new item, hence by Definition 7 it follows that $p_9 \not\lhd_{mst} q_9$ and $q_9 \not\lhd_{mst} p_9$. The fact that $p_9 \not\lhd_{mst} q_9$ is somehow arguable. Actually, Definition 7 is considering the choice between !addItem and !checkOut as a *demonic* non-deterministic choice, and not as a proper internal choice of the client. In the second interpretation of the choice, it would be reasonable to say that p_9 is compliant with q_9, because the client always has the opportunity to terminate. Instead, it is intuitively correct to say q_9 is *not* compliant with p_9, because the store cannot internally choose to terminate the loop and succeed.

Should-testing compliance. We now present a notion of compliance inspired by the theory of should-testing [29,53]. A contract p is *should-testing compliant* with q (in symbols, $p \lhd_{shd} q$) if, after every possible *finite* τ-trace of $p \parallel q$, there exists a subsequent (finite) τ-trace which leads p to the success state.

Definition 8 (Should-testing compliance). *We write* $p \lhd_{shd} q$ *iff*

$$p \parallel q \;\Rightarrow\; p' \parallel q' \quad \textit{implies} \quad \exists q'' \;.\; p' \parallel q' \Rightarrow \mathbf{0} \parallel q''$$

A notion similar to the one in Definition 8 has been used in [26] (under the name of *correct contract composition*), and in [56,9] (where it is named *weak termination*). A stricter notion, called *strong compliance* in [28], also requires that each output is matched by a corresponding input (similarly to *interface automata compatibility* in Definition 11).

Example 9. Consider the pair of contracts (21) in Figure 6. After each (finite) sequence of internal actions of q_{21}, the two contracts can synchronise and reach success. Therefore, $p_{21} \bowtie_{shd} q_{21}$. In the pair (11), even though the two contracts enjoy progress, they can never reach the success state. Hence, we have that $p_{11} \not\lhd_{shd} q_{11}$ and $q_{11} \not\lhd_{shd} p_{11}$.

The previous example shows that, unlike progress, to be should-testing compliant it is not enough that two contracts have infinite interactions, but at any moment, it must be possible for them to succeed.

Example 10. Recall the contracts p_9 and q_9 from Example 8. We have that $p_9 \bowtie_{shd} q_9$, because at each point of the interaction it is possible to make both the client and the online store succeed. As remarked in Example 8, $q_9 \lhd_{shd} p_9$ may be counter-intuitive, because the success of the store relies on the assumption that the client will eventually choose to check out.

The previous example highlights the difference among the three notions of testing-based compliance in Definitions 6 to 8. While in may-testing compliance all choices are angelic and in must-testing compliance they are all demonic, in should-testing compliance there are two kinds of choices. In the first part of the computation, $p \parallel q \Rightarrow p' \parallel q'$ in Definition 8, all the choices are demonic, while in the second part, $p' \parallel q' \Rightarrow \mathbf{0} \parallel q''$, they are all angelic.

None of the notions of compliance studied in this section considers the case where the choices of participant p are angelic, while those of its counterpart q are demonic. The works [11,10] explore this research direction, by interpreting contracts as multi-player concurrent games. However, since the setting is quite different from ours (i.e., the contracts in [11,10] are event structures, while in Section 2 we model them as states of an LTS), we do not include these game-theoretic notions of compliance in this survey.

Behavioural compliance. Definition 9 below formalises in our setting the relation called *behavioural compliance* in [47,49]. A contract p is compliant with q (in symbols, $p \lhd_{beh} q$), if, in every possible τ-reduct $p' \parallel q'$ of $p \parallel q$, two conditions are satisfied: if the reduct is stuck, then p' has reached success; otherwise, if q' alone can produce an infinite τ-trace, then p' must be able to reach success without further synchronisations.

Definition 9 (Behavioural compliance). *We write $p \lhd_{beh} q$ iff:*

$$p \parallel q \;\Rightarrow\; p' \parallel q' \quad \textit{implies} \quad \big(p' \parallel q' \stackrel{\tau}{\not\to} \textit{ implies } p' = \mathbf{0}\big) \;\wedge\; \big(q' \uparrow \textit{ implies } p' \Rightarrow \mathbf{0}\big)$$

Example 11. The pair of contracts (18) in Figure 6. can loop forever with internal actions, but since neither p_{18} nor q_{18} can reach success, we have $p_{18} \not\lhd_{beh} q_{18}$ and $q_{18} \not\lhd_{beh} p_{18}$. For the pair (21), we have that q_{21} can prevent p_{21} from reaching the success state, by following an infinite internal computation: hence, $p_{21} \not\lhd_{beh} q_{21}$. However, $q_{21} \lhd_{beh} p_{21}$, because $q_{21} \parallel p_{21}$ never gets stuck (so, the first condition in Definition 9 holds), and p_{21} cannot perform internal actions (thus satisfying also the second condition).

Compared to must- and should-testing compliance, behavioural compliance allows two contracts to synchronise forever, without ever reaching success: for instance, in the pair (12) of Figure 6, we have $p_{12} \lhd_{beh} q_{12}$, while $p_{12} \not\lhd_{mst} q_{12}$ and $p_{12} \not\lhd_{shd} q_{12}$. Unlike the notion of progress, behavioural compliance does not allow q (i.e., the participant playing the role of server) to produce infinite vacuous interactions: for instance, we have that $p_{21} \lhd_{pg} q_{21}$, but $p_{21} \not\lhd_{beh} q_{21}$.

I/O compliance. In [17], a contract p is considered compliant with q (in symbols, $p \lhd_{io} q$), if, in every possible τ-reduct $p' \parallel q'$ of $p \parallel q$, the weak outputs of p' are included in the weak inputs of q'; further, if p' has no weak outputs but still some weak inputs, then they include the weak outputs of q'.

Definition 10 (I/O compliance). *We write $p \lhd_{io} q$ iff $p \parallel q \Rightarrow p' \parallel q'$ implies:*

$$p'\Downarrow^{!} \subseteq \mathrm{co}(q'\Downarrow^{?}) \;\wedge\; \big((p'\Downarrow^{!} = \emptyset \wedge p'\Downarrow^{?} \neq \emptyset) \implies \emptyset \neq q'\Downarrow^{!} \subseteq \mathrm{co}(p'\Downarrow^{?})\big)$$

Example 12. Consider the pair of contracts (19) in Figure 6. We have that $p_{19}\Downarrow^{!} = \{!\mathsf{a}\} \subseteq \mathrm{co}(q_{19}\Downarrow^{?}) = \mathrm{co}(\{?\mathsf{a}, ?\mathsf{b}\})$. After the synchronisation on a, p_{19} has no more output actions, and one of its input actions ($?\mathsf{c}$) is matched by a weak output of q_{19}. Hence, $p_{19} \lhd_{io} q_{19}$. With similar arguments, we can show that $q_{19} \lhd_{io} p_{19}$. Consider now the pair (16). After the synchronisation on a, and after that q_{16} commits to the $?\mathsf{b}$ branch, we have that one of the weak outputs of p_{16} (i.e., $!\mathsf{c}$) is not matched by any of the inputs of the reduct of q_{16}, and so $p_{16} \not\lhd_{io} q_{16}$. Further, the inputs of the reduct of q_{16} (i.e., $?\mathsf{c}$) do not include the weak co-outputs of the reduct of p_{16}, and so $q_{16} \not\lhd_{io} p_{16}$.

Note that, unlike the other notions of compliance seen so far, in I/O compliance output actions are interpreted differently from input actions. This difference can be understood by interpreting Definition 10 as a game between the two players p and q, similarly to the bisimulation game [54]. The first condition in Definition 10 requires that, if p' wants to do some output (possibly after some τ-moves), then q' must match it with its inputs; the second condition requires that, if p' is *not* going to do any outputs, but she wants to do some input, then q' must be ready (possibly after some τ-moves) to do some output, and q' cannot have outputs other than those accepted by p'.

I/O compliance coincides with progress on synchronous session types (see Theorems 1 and 2). Instead, when considering *asynchronous* session types, where the interaction between participants is mediated by two unbounded FIFO buffers, I/O compliance is equivalent to a notion of compliance based on progress, orphan messages and unspecified receptions [55]. Note that I/O compliance does not completely rule out vacuous infinite interactions, as witnessed by the pair of contracts (21) in Figure 6.

Interface automata compatibility. Definition 11 below formalises in our setting the notion of *compatibility* between interface automata proposed in [37]. A contract p is compliant with q (in symbols, $p \lhd_{ia} q$), if, in every possible τ-reduct $p' \parallel q'$ of $p \parallel q$, the outputs of p' are included in the *immediate* inputs of q'. The

symmetric version of this relation (i.e., $\lhd_{ia} \cap \lhd_{ia}^{-1}$) is equivalent to the notion of compatibility in [37], under the assumptions that there are only two automata, and that all non-internal variables are shared.

Definition 11 (Interface Automata compatibility). *We write $p \lhd_{ia} q$ iff:*

$$p \parallel q \;\Rightarrow\; p' \parallel q' \;\; \textit{and} \;\; p' \xrightarrow{!\mathsf{a}} \;\; \textit{implies} \;\; q' \xrightarrow{?\mathsf{a}}$$

Example 13. Consider the pair of contracts (13) in Figure 6. After the synchronisation on a, the reduct of p_{13} has no more outputs, so $p_{13} \lhd_{ia} q_{13}$ — even though success has not been reached. The contract q_{13} has no outputs, and so $q_{13} \lhd_{ia} p$, for all p. Consider now the pair (17). Even though in all reachable τ-reducts $p' \parallel q'$ of $p_{17} \parallel q_{17}$, the outputs of p' are included in the *weak* inputs of q', in the reduct where p' and q' have synchronised on a but q' has *not* yet performed its internal action, we have that the outputs p' are not matched by the *immediate* inputs of q' (which are empty). Therefore, $p_{17} \not\lhd_{ia} q_{17}$. We also have that $q_{17} \not\lhd_{ia} p_{17}$, because the output $!\mathsf{a}$ in q_{17} is not immediately matched by the input $?\mathsf{a}$ in p_{17}, which is preceded by an internal action.

The relation in Definition 11 differs from the others seen so far in several aspects. First, output actions are interpreted differently from input actions (this features is shared only with I/O compliance). Second, $\lhd_{ia}$ is more sensitive to internal actions: if an input action $?\mathsf{a}$ is preceded by an internal action, then it is not considered to match the output action $!\mathsf{a}$. For instance, the contracts p_{17} and q_{17} in Figure 6 are compliant according to all the relations considered in this Section 3, except $\lhd_{ia}$. Finally, unlike the other compliance relations (except $\lhd_{may}$), $\lhd_{ia}$ does not guarantee progress, as established by Theorem 1. Indeed, since input actions can be neglected, they can be left waiting forever, as in the contracts in (13), (14) and (15) in Figure 6.

4 Comparing compliance relations

The main results of the paper are summarised in Table 1, which establishes relations between the notions of compliance presented in Section 3. The table may be interpreted as follows. Let the metavariable $\mathbb{P}$ range over *sets* of contracts. If $\mathbb{P}$ occurs in row with label $\lhd_i$ and column with label $\lhd_j$ of Table 1, then:

$$\forall p, q \in \mathbb{P} \;:\; p \lhd_i q \implies p \lhd_j q \tag{1}$$

and the reference next to $\mathbb{P}$ points to the theorem where the inclusion is proved. Instead, if $\neg\mathbb{P}(n)$ occurs in row with label $\lhd_i$ and column $\lhd_j$ of the table, then:

$$\exists p, q \in \mathbb{P} \;:\; p \lhd_i q \;\wedge\; p \not\lhd_j q \tag{2}$$

and (n) is the related counterexample, displayed in Figure 6.

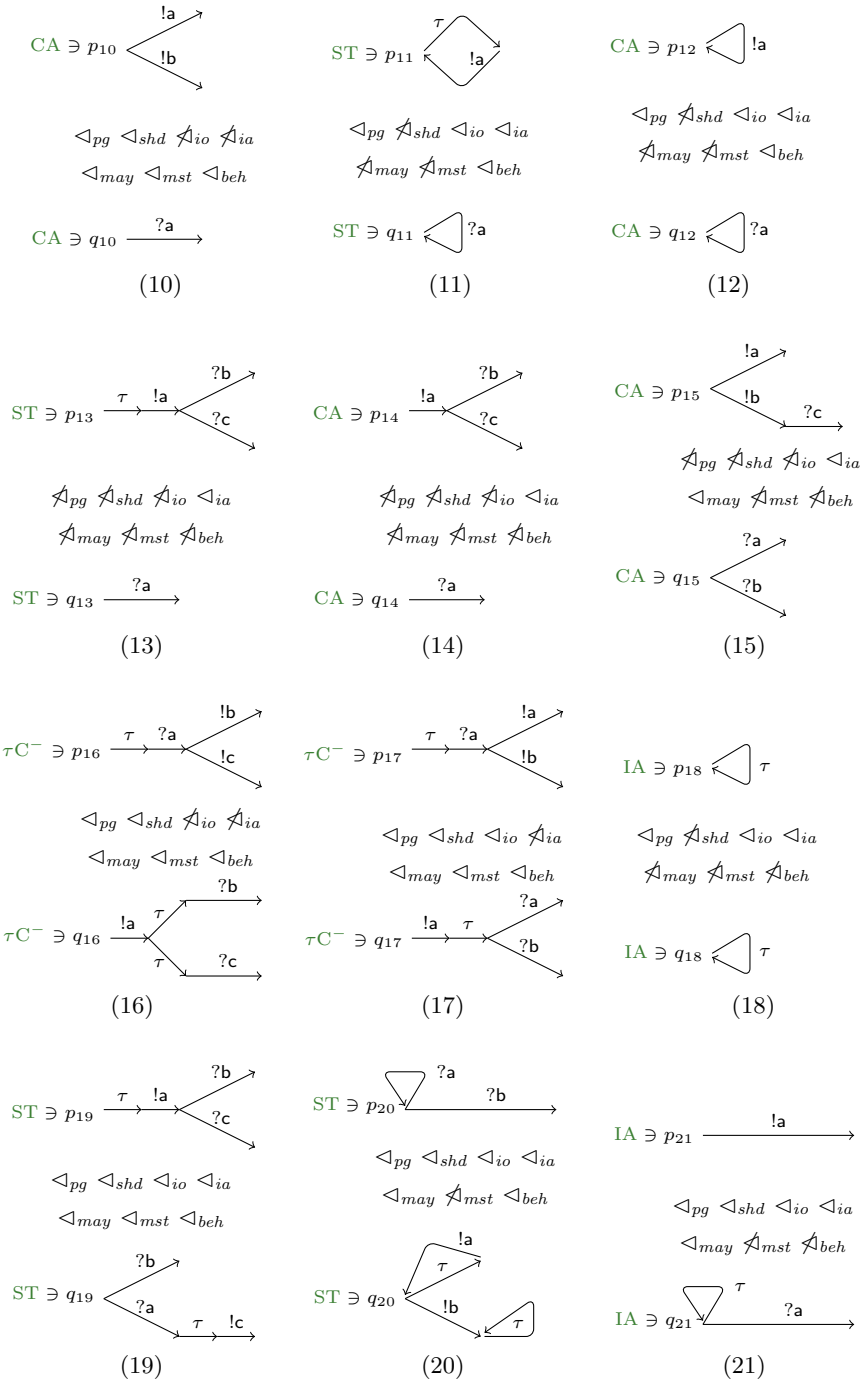


Fig. 6: Some pairs of contracts.

	$\lhd_{pg}$	$\lhd_{shd}$	$\lhd_{io}$	$\lhd_{ia}$	$\lhd_{may}$	$\lhd_{mst}$	$\lhd_{beh}$
$\lhd_{pg}$		$\neg$ST(11) $\neg$CA(12)	ST (th. 2) $\neg$CA(10)	ST (th. 3) $\neg\tau\mathrm{C}^{-}$(16) $\neg$CA(10)	$\neg$ST(11) $\neg$CA(12)	$\neg$ST(11) $\neg$CA(12)	τC (th. 5) $\neg$IA (18)
$\lhd_{shd}$	$\mathbb{U}$ (th. 1)		ST (th. 1 and 2) $\neg$CA(10)	ST (th. 1 and 3) $\neg$CA(10)	$\mathbb{U}$ (th. 1)	$\neg$ST (20)	τC (th. 1 and 5) $\neg$IA (21)
$\lhd_{io}$	$\mathbb{U}$ (th. 1)	$\neg$ST(11) $\neg$CA(12)		$\neg\tau\mathrm{C}^{-}$(17) ST CA (th. 4)	$\neg$ST(11) $\neg$CA(12)	$\neg$ST(11) $\neg$CA(12)	τC (th. 1 and 5) $\neg$IA (18)
$\lhd_{ia}$	$\neg$ST(13) $\neg$CA(14)	$\neg$ST(13) $\neg$CA(14)	$\neg$ST(13) $\neg$CA(14)		$\neg$ST(11) $\neg$CA(12)	$\neg$ST(11) $\neg$CA(12)	$\neg$ST(13) $\neg$CA(14)
$\lhd_{may}$	$\neg$CA(15)	$\neg$CA(15)	$\neg$CA(15)	$\neg$CA(10)		$\neg$CA(15)	$\neg$CA(15)
$\lhd_{mst}$	$\mathbb{U}$ (th. 1)	$\mathbb{U}$ (th. 1)	$\neg$CA(10)	$\neg$CA(10)	$\mathbb{U}$ (th. 1)		$\mathbb{U}$ (th. 1)
$\lhd_{beh}$	$\mathbb{U}$ (th. 1)	$\neg$ST(11) $\neg$CA(12)	$\neg$CA(10)	$\neg$CA(10)	$\neg$ST(11) $\neg$CA(12)	$\neg$ST(11,20) $\neg$CA(12)	

Table 1: Comparison of compliance relations.

Theorem 1. *We have the following inclusions in $\mathbb{U}\times\mathbb{U}$. All the inclusions are strict, and no inclusion exists (in $\mathbb{U}\times\mathbb{U}$) where none is shown.*

$$\begin{array}{ccccc} \lhd_{mst} & \subset & \lhd_{shd} & \subset & \lhd_{may} \\ \cap & & \cap & & \\ \lhd_{beh} & \subset & \lhd_{pg} & \supset & \lhd_{io} \end{array}$$

Proof. The counterexamples to equalities are linked by Table 1. The inclusions in the first row and $\lhd_{beh}\subseteq\lhd_{pg}$ are immediate from Definitions 5 to 9. The inclusion $\lhd_{shd}\subseteq\lhd_{pg}$ follows by Definition 5, since $\lhd_{pg}$ also allows for infinite computations which can never reach success. The inclusion $\lhd_{io}\subseteq\lhd_{pg}$ is direct consequence of Theorem 4.9(a) in [55]. To prove $\lhd_{mst}\subseteq\lhd_{beh}$, assume $p_0\lhd_{mst}q_0$, and let p_k and q_k be such that $p_0\parallel q_0\Rightarrow p_k\parallel q_k$. If $p_k\parallel q_k\overset{\tau}{\not\to}$, then the trace is τ-maximal, and so by $p_0\lhd_{mst}q_0$ it follows that $p_i=\mathbf{0}$ for some $i\le k$. Hence, $p_k=\mathbf{0}$. If $q_k\uparrow$, then we have an infinite τ-maximal trace:

$$p_0\parallel q_0\ \Rightarrow\ p_k\parallel q_k\ \overset{\tau}{\to}\ p_{k+1}\parallel q_{k+1}\ \overset{\tau}{\to}\ p_{k+2}\parallel q_{k+2}\ \overset{\tau}{\to}\ \cdots\ (\forall j\ge 0\,.\,p_{k+j}=p_k)$$

By $p_0\lhd_{mst}q_0$, it follows that $p_i=\mathbf{0}$ for some $i\ge 0$. If $i\le k$, we already have the thesis, since $p_i=p_k=\mathbf{0}$. Otherwise, if $i>k$, then it must be $\mathbf{0}=p_i=p_k$. So, we conclude that $p_0\lhd_{beh}q_0$. □

Theorem 2. $\forall p,q\in\mathrm{ST}\ :\ p\lhd_{pg}q\implies p\lhd_{io}q$

Proof. Direct consequence of Theorem 4.9(b) in [55]. □

Theorem 3. $\forall p,q\in\mathrm{ST}:\ p\lhd_{pg}q\implies p\lhd_{ia}q$

Proof. By contradiction, assume that $p\lhd_{pg}q$ but $p\not\lhd_{ia}q$. Hence, there exists p',q' and a such that $p\parallel q\Rightarrow p'\parallel q'$ with $p'\overset{!\mathsf{a}}{\to}$ and $q'\overset{?\mathsf{a}}{\not\to}$. By Definition 5, $p\lhd_{pg}q$

implies $p' \lhd_{pg} q'$. By definition of ST, it cannot be $q'\!\uparrow$, and the only outgoing transition of p' is $!\mathsf{a}$. Therefore, to have $p' \lhd_{pg} q'$ it must be $q' \,(\xrightarrow{\tau})^n\, q'' \xrightarrow{?\mathsf{a}}$, for some q''. Since $q' \not\xrightarrow{?\mathsf{a}}$, then it must be $n > 0$. This contradicts the definition of ST, which forbids τ immediately before inputs. □

Theorem 4. $\forall p, q \in \mathrm{ST} \cup \mathrm{CA} \;:\; p \lhd_{io} q \implies p \lhd_{ia} q$

Proof. By contradiction, assume that $p \lhd_{io} q$ but $p \not\lhd_{ia} q$. Then, there exist p', q' and a such that $p \parallel q \Rightarrow p' \parallel q'$, with $p' \xrightarrow{!\mathsf{a}}$ and $q' \not\xrightarrow{?\mathsf{a}}$. However, by definition of $\lhd_{io}$, we have $!\mathsf{a} \in \mathrm{co}(q'\Downarrow^{?})$, i.e. $?\mathsf{a} \in q'\Downarrow^{?}$. Since $q \in \mathrm{ST} \cup \mathrm{CA}$, this implies that $?\mathsf{a} \in q'\!\downarrow$, and so $q' \xrightarrow{?\mathsf{a}}$ — contradiction. □

Theorem 5. $\forall p, q \in \tau\mathrm{C} \;:\; p \lhd_{pg} q \implies p \lhd_{beh} q$

Proof. Since $q \in \tau\mathrm{C}$, then by Definition 3 it is not possible that $q\!\uparrow$, hence the thesis follows by Definitions 5 and 9. □

5 Related work and conclusions

We have given a unifying overview of some notions of compliance found in the literature. Although our analysis is still preliminary and partial, as far as we know ours is the first study which systematically organizes compliance relations between behavioural contracts, specified in a general model as arbitrary LTSs.

Previous works survey compliance relations in restricted classes of contracts. The work [24] proposes three notions of compliance for Web services, which are modelled as deterministic contract automata (i.e., there are no internal transitions, and a state cannot have two outgoing transitions with the same label). Two of the proposed notions correspond to $\bowtie_{ia}$ and $\bowtie_{may}$, while the other one requires that, for all reachable states $p' \parallel q'$, it holds $p'\!\downarrow = \mathrm{co}(q'\!\downarrow)$; hence, this relation is stricter than $\bowtie_{io}$. The work [30] surveys some compliance relations for τ-less CCS contracts, while [22] considers *higher-order* session types (i.e., also featuring session delegation). The latter work also studies how the choice of the compliance relation impacts the type systems. Our investigation focuses instead on compliance relations among *arbitrary* LTSs, of which τ-less CCS contracts and (first-order) session types are a special case.

Much work remains to do: besides including other notions of compliance and of contract, we also aim at classifying them according to some relevant criteria: e.g., decidability, computational complexity, *etc.* Another property is whether a compliance relation is preserved when passing from synchronous to asynchronous semantics. For instance, in [17] it is shown that if two session types are I/O compliant, then they will be such also in the presence of asynchrony, i.e. when the communication is mediated by unbounded buffers. This is a relevant property, because it allows to safely approximate an undecidable notion (e.g., compliance between *asynchronous* session types) with a decidable one (e.g., compliance between *synchronous* session types).

The starting point of our investigation about compliance dates back to [12], where the idea of defining a function invocation mechanism based on abidance by behavioural contracts was first developed. In this setting, functions are specified in a λ-calculus enriched with side effects (called *events*), and contracts are automata denoting sets of permitted sequences of events. Calling a function consists in advertising a contract; any function with a behaviour conforming to the contract can be dynamically bound to the callee. This *call-by-contract* mechanism is further studied in [13,14], where techniques are developed to compose untrusted services while guaranteeing to always respect contracts at run-time.

The notion of agreement introduced in [11] is built on an interpretation of contracts as multi-player concurrent games on event structures. A participant (with a given contract) agrees with another participant's contract if she has a *strategy* to interact with the other so that in each interaction she either wins, or it is possible to blame the other participant for not honouring his obligations. A relation between this notion of agreement and progress in session types is shown in [10]: two session types are compliant according to Definition 5 whenever, in their encoding as event structures, *all* innocent strategies of the first participant are winning. We expect that this game-theoretic interpretation of compliance can lead to further correspondence results: for instance, we conjecture that compliance between retractable contracts [5] (which is like progress, but in a semantics which allows some internal choices to be rolled back), corresponds to the existence of a winning *cooperative* strategy in their encodings at event structures. Other notions of compliance which are coarser than progress allow clients to skip some of the messages sent by the server [4], to asynchronously match requests after the corresponding offers have been delivered [19], or they use an external orchestrator with buffering capabilities to suitably rearrange messages [34,6].

In asynchronous models, like e.g. communicating finite-state machines [25] (CFSMs), asynchronous session types [45], and in the choreographies in [27], one has to take into account for vacuous progress due to iterated output and buffering of messages which are never read. Therefore, besides progress, in these asynchronous models compliance usually requires that certain unsafe configurations, like e.g. *orphan messages* and *unspecified receptions*, are not reachable [40]. Compliance in these models is undecidable in general, e.g. the halting problem in Turing machines can be reduced to reachability in CFSMs [25] (decidability only holds under strong restrictions on the general model, e.g. by considering two CFSMs with half-duplex buffers [35]). Algorithmic techniques to safely over-approximate compliance have then been studied, e.g. in [21,50,41,51]. For instance, the results in [50] guarantee that a set of asynchronous session types are compliant whenever it is possible to synthesise a choreography from them. To set asynchronous notions of compliance in our framework, one has to interpret CFSMs / asynchronous session types as contracts in the LTS of Section 2. Some first results in this direction are presented in [55], which shows that for *binary* asynchronous session types, I/O compliance is equivalent to a notion of compliance based on progress, orphan messages and unspecified receptions.

Acknowledgments. We warmly thank Emilio Tuosto and the anonymous reviewers for their insightful comments. This work has been partially supported by Aut. Reg. of Sardinia grants L.R.7/2007 CRP-17285 (TRICS) and P.I.A. 2010 ("Social Glue"), 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. L. Acciai and M. Boreale. A type system for client progress in a service-oriented calculus. In *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, pages 642–658, 2008.
2. L. Acciai, M. Boreale, and G. Zavattaro. Behavioural contracts with request-response operations. In *Proc. COORDINATION*, pages 16–30, 2010.
3. F. Barbanera and U. de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In *Proc. PPDP*, pages 155–164, 2010.
4. F. Barbanera and U. de'Liguoro. Loosening the notions of compliance and sub-behaviour in client/server systems. In *Proc. ICE*, volume 166 of *EPTCS*, pages 94–110, 2014.
5. F. Barbanera, M. Dezani-Ciancaglini, I. Lanese, and U. de'Liguoro. Retractable contracts. In *Proc. PLACES*, EPTCS, 2015. To appear.
6. F. Barbanera, S. van Bakel, and U. de'Liguoro. Orchestrated compliance for session-based client/server interactions. In *Proc. ICE*, EPTCS, 2015. To appear.
7. M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. Compliance and subtyping in timed session types. In *Proc. FORTE*, pages 161–177, 2015.
8. M. Bartoletti, T. Cimoli, and G. M. Pinna. Lending Petri nets and contracts. In *Proc. FSEN*, volume 8161 of *LNCS*, pages 66–82. Springer, 2013.
9. M. Bartoletti, T. Cimoli, and G. M. Pinna. Lending Petri nets. *Science of Computer Programming*, 2015. To appear.
10. M. Bartoletti, T. Cimoli, G. M. Pinna, and R. Zunino. Contracts as games on event structures. *JLAMP*, 2015. (to appear).
11. M. Bartoletti, T. Cimoli, and R. Zunino. A theory of agreements and protection. In *Proc. POST*, volume 7796 of *LNCS*, pages 186–205. Springer, 2013.
12. M. Bartoletti, P. Degano, and G. L. Ferrari. Types and effects for secure service orchestration. In *Proc. IEEE Computer Security Foundations Workshop*, pages 57–69, 2006.
13. M. Bartoletti, P. Degano, and G. L. Ferrari. Planning and verifying service composition. *Journal of Computer Security*, 17(5):799–837, 2009.
14. M. Bartoletti, P. Degano, G. L. Ferrari, and R. Zunino. Semantics-based design for secure Web services. *IEEE Trans. Software Eng.*, 34(1):33–49, 2008.
15. M. Bartoletti, J. Lange, A. Scalas, and R. Zunino. Choreographies in the wild. *Science of Computer Programming*, 2015.
16. M. Bartoletti, A. Scalas, E. Tuosto, and R. Zunino. Honesty by typing. In *Proc. FORTE*, pages 305–320, 2013.
17. M. Bartoletti, A. Scalas, and R. Zunino. A semantic deconstruction of session types. In *Proc. CONCUR*, pages 402–418, 2014.
18. M. Bartoletti, E. Tuosto, and R. Zunino. On the realizability of contracts in dishonest systems. In *Proc. COORDINATION*, pages 245–260, 2012.
19. D. Basile, P. Degano, and G. L. Ferrari. Automata for analysing service contracts. In *Proc. TGC*, pages 34–50, 2014.

20. D. Basile, P. Degano, G. L. Ferrari, and E. Tuosto. From orchestration to choreography through contract automata. In *Proc. ICE*, pages 67–85, 2014.
21. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *Proc. POPL*, pages 191–202, 2012.
22. G. Bernardi, O. Dardha, S. Gay, and D. Kouzapas. On duality relations for session types. In *Proc. TGC*, pages 51–66, 2014.
23. G. Bernardi and M. Hennessy. Compliance and testing preorders differ. In *SEFM Workshops*, 2013.
24. L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are two Web services compatible? In *Proc. TES*, pages 15–28, 2004.
25. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
26. M. Bravetti and G. Zavattaro. Contract based multi-party service composition. In *Proc. FSEN*, volume 4767 of *LNCS*, pages 207–222, 2007.
27. M. Bravetti and G. Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In *Proc. WS-FM*, pages 37–54, 2008.
28. M. Bravetti and G. Zavattaro. A theory of contracts for strong service compliance. *Mathematical Structures in Computer Science*, 19(3):601–638, 2009.
29. E. Brinksma, A. Rensink, and W. Vogler. Fair testing. In *Proc. CONCUR*, pages 313–327, 1995.
30. M. Bugliesi, D. Macedonio, L. Pino, and S. Rossi. Compliance preorders for Web services. In *Proc. WS-FM*, pages 76–91, 2009.
31. M. G. Buscemi and H. C. Melgratti. Contracts for abstract processes in service composition. In *Proc. FIT*, pages 9–27, 2010.
32. S. Carpineti, G. Castagna, C. Laneve, and L. Padovani. A formal account of contracts for Web services. In *Proc. WS-FM*, pages 148–162, 2006.
33. G. Castagna, M. Dezani-Ciancaglini, E. Giachino, and L. Padovani. Foundations of session types. In *Proc. PPDP*, 2009.
34. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for Web services. *ACM TOPLAS*, 31(5):19:1–19:61, 2009.
35. G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
36. M. Coppo, M. Dezani-Ciancaglini, and N. Yoshida. Asynchronous session types and progress for object oriented languages. In *Proc. FMOODS*, pages 1–31, 2007.
37. L. de Alfaro and T. A. Henzinger. Interface automata. In *Proc. ACM SIGSOFT*, pages 109–120, 2001.
38. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.
39. R. De Nicola and M. Hennessy. CCS without tau's. In *Proc. TAPSOFT*, pages 138–152, 1987.
40. P. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *Proc. ESOP*, pages 194–213, 2012.
41. P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *Proc. ICALP*, pages 174–186, 2013.
42. S. Gay and M. Hole. Subtyping for session types in the Pi calculus. *Acta Inf.*, 42(2):191–225, 2005.
43. K. Honda. Types for dyadic interaction. In *Proc. CONCUR*, pages 509–523, 1993.
44. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proc. ESOP*, volume 1381 of *LNCS*, pages 122–138, 1998.

45. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *Proc. POPL*, pages 273–284, 2008.
46. H. Hüttel et al. Foundations of behavioural types. Submitted. Available online at `www.behavioural-types.eu/publications/WG1-State-of-the-Art.pdf`, 2015.
47. C. Laneve and L. Padovani. The *Must* preorder revisited. In *Proc. CONCUR*, pages 212–225, 2007.
48. C. Laneve and L. Padovani. The pairing of contracts and session types. In *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, pages 681–700, 2008.
49. C. Laneve and L. Padovani. An algebraic theory for Web service contracts. *Formal Aspects of Computing*, pages 1–28, 2015.
50. J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *Proc. CONCUR*, pages 225–239, 2012.
51. J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In *Proc. POPL*, pages 221–232, 2015.
52. R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.
53. A. Rensink and W. Vogler. Fair testing. *Information and Computation*, 205(2):125–198, 2007.
54. D. Sangiorgi. *An introduction to bisimulation and coinduction*. Cambridge University Press, Cambridge, UK New York, 2012.
55. A. Scalas. *A semantic deconstruction of session types*. PhD thesis, University of Cagliari, 2015.
56. W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010.