Orchestrated Session Compliance [*]

Franco Barbanera
Dipartimento di Matematica e Informatica
University of Catania
barba@dmi.unict.it

Steffen van Bakel
Department of Computing
Imperial College London
svanbakel@imperial.ac.uk

Ugo de'Liguoro
Dipartimento di Informatica
University of Torino
ugo.deliguoro@unito.it

We investigate the notion of orchestrated compliance for client/server interactions in the context of *session contracts*. Devising the notion of orchestrator in such a context makes it possible to have orchestrators with unbounded buffering capabilities and at the same time to guarantee any message from the client to be eventually delivered by the orchestrator to the server, while preventing the server from sending messages which are kept indefinitely inside the orchestrator. The compliance relation is shown to be decidable by means of *1*) a procedure synthesising the orchestrators, if any, making a client compliant with a server, and *2*) a procedure for deciding whether an orchestrator behaves in a proper way as mentioned before.

1 Introduction

Session types and contracts are two formalisms used to study client/server protocols. Session types have been introduced in [16] as a tool for statically checking safe message exchanges through channels. Contracts, on the other hand, as proposed in [12, 18, 13], are a subset of CCS without τ, that address the problem of abstractly describing behavioural properties of systems by means of process algebra. In between these two formalisms lie *session contracts*[1] as introduced in [1, 3, 8, 9]; this is a formalism interpreting the session types into a subset of contracts.

In the theory of contracts, as well as in the formalism of session contracts, the notion of *compliance* plays a central role. A client ρ is defined as being compliant with a server σ (written as $\rho \dashv \sigma$) whenever *all* of its *requests* are satisfied by the server. Now it might be the case that client satisfaction cannot be achieved just because of a difference in the order in which the partners exchange information, or because one of them provide some extra un-needed information.

Consider the example of a meteorological data processing system (MDPS) that is permanently connected to a weather station to which it sends, according to its processing needs, particular data requests. For the sake of simplicity, we consider just two particular requests, namely for *temperature* and *humidity*. After the requests, the MDPS expects to receive the data in the order they were requested. In the session-contracts formalism the interface for the simplified MDPS can be stated as follows:

$$\mathsf{MDPS} \;=\; \mathtt{rec}\,x.\overline{\mathtt{tempReq}}.\overline{\mathtt{humReq}}.\mathtt{temperature}.\mathtt{humidity}.x$$

(Here, as in CCS, a symbol like '$\mathtt{a}$' stands for on input action, whereas '$\overline{\mathtt{a}}$' denotes the corresponding output). We assume a weather station to be able to send back the asked-for information in the order

[*] This work was partially supported by COST Action IC1201 BETTY, MIUR PRIN Project CINA Prot. 2010LHT4KM and Torino University/Compagnia San Paolo Project SALT.

[1] They were dubbed *session behaviours* in [1, 3]. For sake of uniformity and since *session contract* sounds more appealing, we adhere here to this name.

S. Knight, I. Lanese, A. Lluch Lafuente and H. T. Vieira (Eds.):
8th Interaction and Concurrency Experience (ICE 2015)
EPTCS 189, 2015, pp. 21–36, doi:10.4204/EPTCS.189.4

decided by its sensors, interspersed with information about *wind speed*:

$$\begin{array}{rcl}\mathsf{WeatherStation} & = & \mathsf{rec}\,x.\mathtt{tempReq}.\mathtt{humReq}.(\overline{\mathtt{temperature}}.\overline{\mathtt{humidity}}.\overline{\mathtt{wind}}.x \\ & & \qquad\qquad\qquad\qquad\quad \oplus \\ & & \qquad\qquad\qquad\qquad\quad \overline{\mathtt{humidity}}.\overline{\mathtt{temperature}}.\overline{\mathtt{wind}}.x)\end{array}$$

With the standard notion of compliance, it is not difficult to check that $\mathsf{MDPS} \not\dashv \mathsf{WeatherStation}$, since the client MDPS has no input action for the wind data, and also since it could occur that the temperature and humidity data are delivered in a different order than expected by the MDPS.

A natural solution to this would consist of devising a process that acts as a mediator (here called *orchestrator*) between the client and the server, coordinating them in a centralised way in order to make them compliant. This sort of solution is the one adopted in the practice of web-service interaction, in particular for business processes, where the notion of orchestration has been introduced and developed:

> “ Orchestration*: Refers to an executable business process that may interact with both internal and external web services. Orchestration describes how web services can interact at the message level, including the business logic and execution order of the interactions.* ” [20]

In the context of the theory of contracts, this solution was formalised and investigated by Padovani [19], where orchestrators are processes that cannot affect the internal decisions of the client nor of the server, but can affect the way their synchronisation is carried out.

The orchestrating actions an orchestrator can perform have the following forms:

$\langle \mathtt{a},\overline{\mathtt{a}}\rangle$ (resp. $\langle \overline{\mathtt{a}},\mathtt{a}\rangle$): the orchestrator gets $\mathtt{a}$ from the client (resp. server) and immediately delivers it to the server (resp. client) in a synchronous way.

$\langle \mathtt{a},\varepsilon\rangle$ (resp. $\langle \varepsilon,\mathtt{a}\rangle$): the orchestrator gets $\mathtt{a}$ from the client (resp. server) and stores it in the buffer.

$\langle \overline{\mathtt{a}},\varepsilon\rangle$ (resp. $\langle \varepsilon,\overline{\mathtt{a}}\rangle$): the orchestrator takes $\mathtt{a}$ from the buffer and sends it to the client (resp. server).

So a possible orchestrator enabling compliance for our example would be

$$\begin{array}{rcl}\mathsf{f} & = & \mathsf{rec}\,x.\langle \mathtt{tR},\overline{\mathtt{tR}}\rangle.\langle \mathtt{hR},\overline{\mathtt{hR}}\rangle.(\langle\overline{\mathtt{t}},\mathtt{t}\rangle.\langle\overline{\mathtt{h}},\mathtt{h}\rangle.\langle\varepsilon,\mathtt{w}\rangle.x \\ & & \qquad\qquad\qquad\qquad\quad \vee \\ & & \qquad\qquad\qquad\qquad\quad \langle\varepsilon,\mathtt{h}\rangle.\langle\overline{\mathtt{t}},\mathtt{t}\rangle.\langle\overline{\mathtt{h}},\varepsilon\rangle.\langle\varepsilon,\mathtt{w}\rangle.x)\end{array}$$

where $\mathtt{tR}$, $\mathtt{hR}$, $\mathtt{t}$, $\mathtt{h}$, and $\mathtt{w}$ stand for $\mathtt{tempReq}$, $\mathtt{humReq}$, $\mathtt{temperature}$, $\mathtt{humidity}$, and $\mathtt{wind}$, respectively. The orchestrator f rearranges the order of messages when necessary, and *retains* the wind information, not needed by MDPS.

Actually, the orchestrator f is not a valid orchestrator in the sense of [19]: indeed the $\mathtt{wind}$ information is never delivered to the client (i.e. it is implicitly discarded), so that the buffer corresponding to f would be unbounded. Unbounded buffers are not allowed in [19], where boundedness of buffers is used to guarantee both decidability and the possibility of synthesising orchestrators. In a session setting instead, as is the present one, decidability and orchestrators synthesis can be established even in presence of unbounded buffering capabilities of orchestrators.

In a two-parties session-based interaction, the choice among several continuations always depends on just one of the two actors. To let our formalism fully adhere to such a viewpoint our *session orchestrators*, besides (as argued in [19]) being processes that cannot affect the internal decision of the client or the server, are such that they do not create any non-determinism besides that already present in the partners. This will correspond to restricting the syntax in such a way that orchestrators like, for instance, $\langle\varepsilon,\mathtt{a}\rangle.\mathsf{f}_1 \vee \langle\mathtt{b},\varepsilon\rangle.\mathsf{f}_2$, are not allowed. In fact, in the latter orchestrator, the choice of receiving an input from the client or from the server would not depend solely on the partners. The f described above does respect this syntax restriction.

$$\begin{array}{rcll} \sigma,\rho & ::= & \mathbf{1} & \text{success} \\ & \mid & a_1.\sigma_1 + \cdots + a_n.\sigma_n & \text{external choice} \\ & \mid & \overline{a}_1.\sigma_1 \oplus \cdots \oplus \overline{a}_n.\sigma_n & \text{internal choice} \\ & \mid & x & \text{variable} \\ & \mid & \mathsf{rec}\, x\,.\,\sigma & \text{recursion} \end{array}$$

Figure 1: The grammar of raw session-contracts

Moreover, in our system it will be possible to prove that $\mathsf{f} : \mathsf{MDPS} \dashv\mid \mathsf{WeatherStation}$ i.e.: MDPS and WeatherStation manage to be compliant (represented by $\dashv\mid$ in our context) when their interaction is mediated by f. In our system we will also manage to prevent the presence of *fake orchestrated complying interactions*, like that between the client $\overline{\mathtt{a}}.\overline{\mathtt{b}}$ and the server $\mathtt{a}$ through the orchestrator $\langle \mathtt{a}, \overline{\mathtt{a}} \rangle.\langle \mathtt{b}, \varepsilon \rangle$. In this case the client gets the illusion that *all its requests* are satisfied, whereas its output b never reaches the server, but will be indefinitely kept *inside* the orchestrator's buffer. While in the contract setting of [19] such compliant interactions are allowed, in our session context we manage to rule out orchestrators behaving like $\langle \mathtt{a}, \overline{\mathtt{a}} \rangle.\langle \mathtt{b}, \varepsilon \rangle$, which never deliver a message from the client to the server.

We shall prove that properties like the one just mentioned, characterising *well-behaved* orchestrators, are decidable. Given an f, decidability of orchestrated compliance through f will be proved. We will also show that, given a client and a server, it is possible to synthesise *all* the orchestrators that make the client and system compliant, if any.

2 Session contracts and orchestrated compliance

Session contracts are a restriction of *contracts* [18, 13]. They are designed to be in one-to-one correspondence to session types [16] without delegation (in [1, 3] a version with delegation was investigated). The restriction consists in constraining internal and external choices in a way that limits the non-determinism to (internal) output selection.

Definition 2.1 (Session Contracts)**.** *i) Let $\mathcal{N}$ be a countable set of symbols and $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$. The set* RSC *of* raw session contracts *is defined by the grammar in Figure 1, where:*

- *for external and internal choices, $n \geq 1$, and $a_i \in \mathcal{N}$ (hence $\overline{a}_i \in \overline{\mathcal{N}}$) for all $1 \leq i \leq n$;*
- *the variable x is a* session-contract variable *out of a denumerable set; we consider occurrences of x in σ* bound *in $\mathsf{rec}\, x\,.\,\sigma$. An occurrence of x in σ is* free *if it is not bound, and we write $\mathrm{FV}(\sigma)$ for the set of free variables in σ. σ is said to be* closed *whenever $\mathrm{FV}(\sigma) = \emptyset$.*

Act $= \mathcal{N} \cup \overline{\mathcal{N}}$ *is the set of* actions.

ii) The set SC *of* **session contracts** *is the subset of closed raw session contracts such that in $a_1.\sigma_1 + \cdots + a_n.\sigma_n$ and $\overline{a}_1.\sigma_1 \oplus \cdots \oplus \overline{a}_n.\sigma_n$, the a_i and the $\overline{a}_i$ are, in both, pairwise distinct; moreover, in $\mathsf{rec}\, x\,.\,\sigma$ the expression σ is not a variable.*

As usual, we abbreviate $a_1.\sigma_1 + \cdots + a_n.\sigma_n$ by $\sum_{i=1}^{n} a_i.\sigma_i$, and $\overline{a}_1.\sigma_1 \oplus \cdots \oplus \overline{a}_n.\sigma_n$ by $\bigoplus_{i=1}^{n} \overline{a}_i.\sigma_i$. We also use the notations $\sum_{i \in I} a_i.\sigma_i$ and $\bigoplus_{i \in I} \overline{a}_i.\sigma_i$ for finite and non-empty I. We take the equi-recursive view of recursion, by equating $\mathsf{rec}\, x\,.\,\sigma$ with $\sigma\{x/\mathsf{rec}\, x\,.\,\sigma\}$.

The trailing $\mathbf{1}$ is normally omitted: for example, we will write $a + b$ for $a.\mathbf{1} + b.\mathbf{1}$. Session contracts will be considered modulo commutativity of internal and external choices.

The operational semantics of session contracts is given in terms of a labeled transition system (LTS) $\sigma \xrightarrow{\alpha} \sigma'$ where $\sigma, \sigma' \in \mathsf{SC}$ and α either belongs to a set of actions **Act** or is an internal action τ.

Definition 2.2 (LTS for Session Contracts)**.** *We define the labelled transition system* $(\mathsf{SC}, \mathbf{Act}, \rightarrow)$ *by*

$$\overline{a}_1.\sigma_1 \oplus \cdots \oplus \overline{a}_n.\sigma_n \xrightarrow{\tau} \overline{a}_k.\sigma_k \qquad \overline{a}.\sigma \xrightarrow{\overline{a}} \sigma \qquad a_1.\sigma_1 + \cdots + a_n.\sigma_n \xrightarrow{a_k} \sigma_k$$

where $1 \leq k \leq n$*, and* $\sigma \xrightarrow{\alpha} \sigma'$ *is short for* $(\sigma, \alpha, \sigma') \in \rightarrow$*. We shall use* $\rightarrow$ *as shorthand for* $\xrightarrow{\tau}$*. As usual, we write* $\Longrightarrow$ *for* $\rightarrow^*$ *and* $\overset{\alpha}{\Longrightarrow}$ *for* $\rightarrow^* \xrightarrow{\alpha} \rightarrow^*$ *with* $\alpha \in \mathbf{Act}$*.*

Notice that reduction is not defined through contextual rules, so reduction only takes place at the ‘top’ level. Thereby, it is impossible for $\mathsf{rec}\,x.\sigma$ to unfold more than once without consuming a guard (remember that σ is not a variable): so recursion is contractive in the usual sense. We will safely assume that no two consecutive rec binders (as in $\mathsf{rec}\,x.\mathsf{rec}\,y.\sigma$) are present in a session contract.

We observe that $\overset{\alpha}{\Longrightarrow}$ is well defined, in that if $\sigma \in \mathsf{SC}$ and $\sigma \overset{\alpha}{\Longrightarrow} \sigma'$ (or $\sigma \Longrightarrow \sigma'$), then $\sigma' \in \mathsf{SC}$.

Session orchestrators As also done in [19] in the context of the theory of contracts, we intend to investigate the notion of compliance when the interaction between a client and a server is mediated by an *orchestrator*. Different from the broad contract setting, the session setting we are in induces some natural restrictions to the syntax of orchestrators, making it safe to have orchestrators with unbounded buffers. Moreover, it is possible to check whether any output from the client is eventually delivered by the orchestrator to the server, as well as whether there might be an infinite interaction which falsely progresses because it is made only of outputs from the server to the orchestrator (see Section 4).

The set of actions an orchestrator can perform, that we take from [19], have been informally described in the introduction.[2]

It can be reasonably argued that orchestrators must not show any internal non-determinism. Taking into account now the *session-based* interactions of our setting, such an assumption should be further extended, keeping in mind that in a session-based client/server interaction any possible non-determinism is due only to the internal non-determinism of the two partners. We therefore define our *session-orchestrators* so as to enforce this point of view. It follows that the only choice we allow in session-orchestrators (represented by ‘$\vee$’ in expressions like $f \vee g$) is an external one, and it is necessarily driven by the internal choice of one of the two partners. This implies that the actions immediately exhibited by f and g in an orchestrator like $f \vee g$ must have the same *direction*, i.e. must belong to just one of the two subsets $\{\langle a, \varepsilon\rangle, \langle a, \overline{a}\rangle \mid a \in \mathcal{N}\}$ or $\{\langle \varepsilon, a\rangle \mid \langle \overline{a}, a\rangle \mid a \in \mathcal{N}\}$. Besides, orchestration actions of the form $\langle \overline{a}, \varepsilon\rangle$ or $\langle \varepsilon, \overline{a}\rangle$ must be used just as prefixes μ in orchestrators like $\mu.f$. The other ruled-out cases, like $\langle \overline{c}, \varepsilon\rangle.f' \vee \langle \varepsilon, b\rangle.g'$ or $\langle c, \varepsilon\rangle.f' \vee \langle \varepsilon, b\rangle.g'$, would conflict with the session viewpoint or, like $\langle \overline{c}, \varepsilon\rangle.f' \vee \langle b, \varepsilon\rangle.g'$, would be meaningless according to the syntax of session contracts.

We now formally define orchestration actions by partitioning them into different syntactic categories.

Definition 2.3 (Session-orchestration actions)**.** *We define* **OrchAct** *as the set of* session-orchestration actions μ *described by the following grammar (where* $a \in \mathcal{N}$ *and* $\overline{a} \in \overline{\mathcal{N}}$*):*

$$\begin{array}{lllllll} \mu & ::= & \iota_L \mid \iota_R \mid o & \qquad & o & ::= & \langle \overline{a}, \varepsilon\rangle \mid \langle \varepsilon, \overline{a}\rangle \\ \iota_L & ::= & \langle a, \varepsilon\rangle \mid \langle a, \overline{a}\rangle & & \iota_R & ::= & \langle \varepsilon, a\rangle \mid \langle \overline{a}, a\rangle \end{array}$$

[2] One could wonder whether just asynchronous orchestration actions can be taken into account, since any $\langle a, \overline{a}\rangle$ action can be safely mimicked by two asynchronous ones, namely $\langle a, \varepsilon\rangle.\langle \varepsilon, \overline{a}\rangle$ (similarly for $\langle \overline{a}, a\rangle$). A difference in fact would arise only for what concerns implementation, since the protocol for a synchronous exchange would not involve the use of a buffer, which is instead necessary for asynchronous actions. Such an implementation issue seems unlikely to be related to our theoretical treatment. In contrast, we shall point out in Remark 4.5 how implementation related aspects might affect our formalisation.

We let $\mu,\mu',\mu_1,\ldots$ range over orchestration actions, and $\boldsymbol{\mu}$ over both finite sequence $\mu_1\ldots\mu_n$ in $\mathbf{OrchAct}^*$ and infinite sequence $\mu_1\ldots\mu_n\ldots$ in $\mathbf{OrchAct}^\infty$.

Definition 2.4 (Session Orchestrators)**.** *We define* Orch *as the set of* session orchestrators, *ranged over by* $f,g,h,\ldots$, *described by the* closed *terms generated by the following grammar:*

$$
\begin{array}{rcll}
f,g & ::= & \mathtt{1} & \\
 & | & \iota_L.f_1 \vee \cdots \vee \iota_L.f_n & (n \geq 1) \\
 & | & \iota_R.f_1 \vee \cdots \vee \iota_R.f_n & (n \geq 1) \\
 & | & o.f & \\
 & | & x & \\
 & | & \mathsf{rec}\,x.\,f &
\end{array}
$$

We impose session orchestrators to be contractive, *i.e. the* f *in* $\mathsf{rec}\,x.\,f$ *is assumed to not be a variable.*

The expression $\mathtt{1}$ represents the orchestrator offering no action. $o.f$ offers just the orchestration action of the category o and continues as f, whereas $\iota_L.f_1 \vee \cdots \vee \iota_L.f_n$ and $\iota_R.f_1 \vee \cdots \vee \iota_R.f_n$ offer n (uni-directional) actions of the syntactical categories, respectively, ι_L and ι_R. Recursive orchestrators can be expressed by means of the rec binder and recursion variables, in the usual way. As for session contracts, orchestrators are defined as to have recursion variables guarded by at least one orchestration action. In the following we shall often refer to 'session orchestrators' as simply 'orchestrators.' As for session contracts, we take an equi-recursive point of view, so identify $\mathsf{rec}\,x.\,f$ and $f\{x/\mathsf{rec}\,x.\,f\}$.

We now define the operational semantics of orchestrators as an LTS.

Definition 2.5 (LTS for Orchestrators)**.** *We define the labelled transition system* $(\mathsf{Orch}, \mathbf{OrchAct}, \mapsto)$ *by*

$$
\frac{}{\mu.f \xmapsto{\mu} f}
\qquad\qquad
\frac{f \xmapsto{\mu} f'}{f \vee g \xmapsto{\mu} f'}
\qquad\qquad
\frac{g \xmapsto{\mu} g'}{f \vee g \xmapsto{\mu} g'}
$$

Given a sequence $\boldsymbol{\mu}$, *we write* $f \xmapsto{\boldsymbol{\mu}}$ *whenever* $f \xmapsto{\mu_1} f_1 \xmapsto{\mu_2} \cdots \xmapsto{\mu_n} f_n$ *if* $\boldsymbol{\mu} = \mu_1 \cdots \mu_n \in \mathbf{OrchAct}^*$, *or* $f \xmapsto{\mu_1} \cdots \xmapsto{\mu_n} f_n \xmapsto{\mu_{n+1}} \cdots$ *if* $\boldsymbol{\mu} = \mu_1 \cdots \mu_n \cdots \in \mathbf{OrchAct}^\infty$. *We write* $f \not\mapsto$ *if there is no* μ *such that* $f \xmapsto{\mu}$.

Definition 2.6 (Orchestrator Traces)**.** *Let* $f \in \mathsf{Orch}$.

1. *The set* $\mathsf{Tr}(f) \subseteq (\mathbf{OrchAct}^* \cup \mathbf{OrchAct}^\infty)$ *of* traces of f *is defined by:* $\mathsf{Tr}(f) = \{\, \boldsymbol{\mu} \mid f \xmapsto{\boldsymbol{\mu}} \,\}$.
2. *The set* $\mathsf{MaxTr}(f) \subseteq (\mathbf{OrchAct}^* \cup \mathbf{OrchAct}^\infty)$ *of* maximal traces of f *is defined by*

$$
\mathsf{MaxTr}(f) \;=\; \{\boldsymbol{\mu} \in \mathsf{Tr}(f) \mid \exists f'\, [\, f \xmapsto{\boldsymbol{\mu}} f' \not\mapsto \,] \text{ or } \mu \in \mathbf{OrchAct}^\infty\}
$$

As in [19], we define an *orchestrated system* as a triple $\langle \rho, f, \sigma \rangle$ (written $\rho \parallel_f \sigma$) representing ρ (the client) and σ (the server) interacting with each other under the supervision of f.

Definition 2.7 (Orchestrated Systems operational semantics)**.** *The operational semantics of orchestrated systems is defined as follows:*

$$
\frac{\rho \to \rho'}{\rho \parallel_f \sigma \to \rho' \parallel_f \sigma}
\qquad\qquad
\frac{\sigma \to \sigma'}{\rho \parallel_f \sigma \to \rho \parallel_f \sigma'}
$$

$$
\frac{\rho \xrightarrow{\alpha} \rho' \quad f \xmapsto{\langle \overline{\alpha},\alpha \rangle} f' \quad \sigma \xrightarrow{\overline{\alpha}} \sigma'}{\rho \parallel_f \sigma \xrightarrow{\langle \overline{\alpha},\alpha \rangle} \rho' \parallel_{f'} \sigma'}
\qquad
\frac{\rho \xrightarrow{\overline{\alpha}} \rho' \quad f \xmapsto{\langle \alpha,\varepsilon \rangle} f'}{\rho \parallel_f \sigma \xrightarrow{\langle \alpha,\varepsilon \rangle} \rho' \parallel_{f'} \sigma}
\qquad
\frac{f \xmapsto{\langle \varepsilon,\alpha \rangle} f' \quad \sigma \xrightarrow{\overline{\alpha}} \sigma'}{\rho \parallel_f \sigma \xrightarrow{\langle \varepsilon,\alpha \rangle} \rho \parallel_{f'} \sigma'}
$$

We write $\xRightarrow{\mu}$ *for* $\to^* \circ \xrightarrow{\mu} \circ \to^*$, *and* $\xRightarrow{\boldsymbol{\mu}}$ *for* $\xRightarrow{\mu_1} \circ \cdots \circ \xRightarrow{\mu_n}$ *(resp.* $\xRightarrow{\mu_1} \circ \xRightarrow{\mu_2} \circ \cdots$*) if* $\boldsymbol{\mu}$ *is finite (resp. infinite). The notation* $\rho \parallel_f \sigma \not\to$ *will be used when both* $\rho \parallel_f \sigma \not\to$ *(according to the first two rules above) and* $\neg\exists\mu\; [\rho \parallel_f \sigma \xRightarrow{\mu}\;]$ *hold.*

Notice that for the operational semantics of orchestrated systems we have defined labelled reductions instead of a reduction relation (as done in [19]). We label orchestrated-systems' transitions by the orchestration actions which make them possible, since in our setting we need to check for particular conditions of orchestrator buffers after the evolution of an orchestrated system. A buffer can be explicitly coupled with an orchestrator or can be represented implicitly by the actions performed by the orchestrator. The latter is the choice of [19], that we maintain.

We now define a notion of compliance which is coarser than expected because of possible unfair behaviour of the orchestrators, which will be refined in Section 4.

Definition 2.8 (Disrespectful and Strict Orchestrated Compliance)**.** *An orchestrator f is said to be ρ-σ* strict *whenever, for any finite $\boldsymbol{\mu}$, $f \overset{\boldsymbol{\mu}}{\longmapsto}$ implies $\rho \|_f \sigma \overset{\boldsymbol{\mu}}{\Longrightarrow}$. We define:*

i) $f : \rho \dashv\!\!|^{\mathsf{ds}} \sigma$ *if f is ρ-σ strict, and for any $\boldsymbol{\mu}$, ρ' and σ', the following holds:*

$$\rho \|_f \sigma \overset{\boldsymbol{\mu}}{\Longrightarrow} \rho' \|_{f'} \sigma' \not\longrightarrow \quad \textit{implies} \quad \rho' = \mathbf{1}.$$

ii) $\rho \dashv\!\!|^{\mathsf{ds}} \sigma$ *if* $\exists f\, [\, f : \rho \dashv\!\!|^{\mathsf{ds}} \sigma\,]$.

3 Orchestrators Synthesis

In this section we define an inference system $\rhd^{\mathsf{inf}}$ for (possibly open) orchestrators, deducing judgments like $f : \rho \dashv^{\mathsf{ds}} \sigma$, under finitely many assumptions of a certain shape. We first establish that the system is sound with respect to the $\dashv\!\!|^{\mathsf{ds}}$ relation. Then, on the basis of that system, we provide an algorithm **Synth** for orchestrator synthesis which, given ρ and σ, returns the set of all the relevant orchestrators f such that $\rhd^{\mathsf{inf}}\ f : \rho \dashv^{\mathsf{ds}} \sigma$ (namely with $\Gamma = \emptyset$) and hence that $f : \rho \dashv\!\!|^{\mathsf{ds}} \sigma$. The algorithm is essentially an exhaustive proof search for $\rhd^{\mathsf{inf}}$ that can be shown to be always terminating.

Definition 3.1 (The orchestrators inference system $\rhd^{\mathsf{inf}}$)**.** *The judgements of the system are expressions of the form $\Gamma \rhd^{\mathsf{inf}} f : \rho \dashv^{\mathsf{ds}} \sigma$, where $\rho, \sigma \in \mathsf{SC}$, f is a (possibly open) orchestrator and Γ is a set of assumptions of the form $x : \rho_i \dashv^{\mathsf{ds}} \sigma_i$ such that: $x{:}\rho \dashv^{\mathsf{ds}} \sigma \in \Gamma$ & $y{:}\rho \dashv^{\mathsf{ds}} \sigma \in \Gamma \implies x = y$ (so Γ represents an injective mapping from variables to expressions of the form $\rho \dashv^{\mathsf{ds}} \sigma$). The axioms and rules of the system are described in Figure 2.*

In the inference system of Figure 2 the symbol $\dashv^{\mathsf{ds}}$ is a relation symbol representing the relation $\dashv\!\!|^{\mathsf{ds}}$ as defined in Definition 2.8. In order to give the intuition behind the inference system, let us briefly comment on one of the rules, say $(\textsc{Cpl}\Sigma\text{-L})$. In case it is possible to show that f' is an orchestrator for $f_p \dashv\!\!|^{\mathsf{ds}} \sigma$, orchestrated compliance can be obtained for $\sum_{i\in I} a_i.\rho_i \dashv\!\!|^{\mathsf{ds}} \sigma$ by means of $\langle \overline{a}_p, \varepsilon\rangle.f'$, since the $\langle \overline{a}_p, \varepsilon\rangle$ action satisfies one of the *input requests* a_is. In case $x \notin fn(f')$, we get that $\mathsf{rec}\,x.\langle \overline{a}_p, \varepsilon\rangle.f' = \langle \overline{a}_p, \varepsilon\rangle.f'$. This means that axiom $(\textsc{Ax})$ has been used in the derivation of f' and the interaction between $\sum_{i\in I} a_i.\rho_i$ and σ finitely succeeds if the actions described in the branch from $(\textsc{Cpl}\Sigma\text{-L})$ to $(\textsc{Ax})$ are performed. In case $x \in fn(f')$, rule $(\textsc{Hyp})$ has been used for f', and a successful infinite interaction is possible between $\sum_{i\in I} a_i.\rho_i$ and σ when the orchestrator repeatedly performs the actions in the branch from $(\textsc{Cpl}\Sigma\text{-L})$ to $(\textsc{Hyp})$, as described by the recursive orchestrator $\mathsf{rec}\,x.\langle \overline{a}_p, \varepsilon\rangle.f'$.

Definition 3.2 (Judgment Semantics)**.** *Let $\Gamma = \{x_1{:}\rho_1 \dashv^{\mathsf{ds}} \sigma_1, \ldots, x_k{:}\rho_k \dashv^{\mathsf{ds}} \sigma_k\}$, and θ be a map such that $\theta(x_i) = f_i$, where the f_is are proper (i.e. closed) orchestrators. Then we define:*

$$\begin{array}{lcl} \theta \models \Gamma & \triangleq & \forall (x_i{:}\rho_i \dashv^{\mathsf{ds}} \sigma_i) \in \Gamma\, [\, \theta(x_i) : \rho_i \dashv\!\!|^{\mathsf{ds}} \sigma_i\,] \\ \Gamma \models f : \rho \dashv^{\mathsf{ds}} \sigma & \triangleq & \forall \theta\, [\, \theta \models \Gamma \implies \theta(f) : \rho \dashv\!\!|^{\mathsf{ds}} \sigma\,] \end{array}$$

where $\theta(f)$ is the result of substituting, for all variables $x \in f$, all free occurrences of x by $\theta(x)$.

$$(\textsc{Ax}):\ \frac{}{\Gamma \rhd^{\mathsf{inf}} \mathtt{1} : \mathbf{1} \dashv^{\mathsf{cs}} \sigma} \qquad (\textsc{Hyp}):\ \frac{}{\Gamma, x{:}\rho \dashv^{\mathsf{cs}} \sigma \rhd^{\mathsf{inf}} x : \rho \dashv^{\mathsf{cs}} \sigma}$$

$$(\textsc{Cpl}\Sigma\text{-L}):\ \frac{\Gamma,\ x{:}\sum_{i\in I} a_i.\rho_i \dashv^{\mathsf{cs}} \sigma \rhd^{\mathsf{inf}} f' : \rho_p \dashv^{\mathsf{cs}} \sigma}{\Gamma \rhd^{\mathsf{inf}} \mathsf{rec}\, x.\langle \overline{a}_p, \varepsilon\rangle.f' : \sum_{i\in I} a_i.\rho_i \dashv^{\mathsf{cs}} \sigma}\ (p \in I) \qquad (\textsc{Cpl}\Sigma\text{-R}):\ \frac{\Gamma,\ x{:}\rho \dashv^{\mathsf{cs}} \sum_{i\in I} a_i.\sigma_i \rhd^{\mathsf{inf}} f' : \rho \dashv^{\mathsf{cs}} \sigma_p}{\Gamma \rhd^{\mathsf{inf}} \mathsf{rec}\, x.\langle \varepsilon, \overline{a}_p\rangle.f' : \rho \dashv^{\mathsf{cs}} \sum_{i\in I} a_i.\sigma_i}\ (p \in I)$$

$$(\textsc{Cpl}\bigoplus\text{-R}):\ \frac{\Gamma,\ x{:}\bigoplus_{i\in I} \overline{a}_i.\rho_i \dashv^{\mathsf{cs}} \bigoplus_{j\in J} \overline{b}_j.\sigma_j \rhd^{\mathsf{inf}} f_j : \bigoplus_{i\in I} \overline{a}_i.\rho_i \dashv^{\mathsf{cs}} \sigma_j \quad (\forall j \in J)}{\Gamma \rhd^{\mathsf{inf}} \mathsf{rec}\, x.\bigvee_{j\in J}\langle \varepsilon, b_j\rangle.f_j : \bigoplus_{i\in I} \overline{a}_i.\rho_i \dashv^{\mathsf{cs}} \bigoplus_{j\in J} \overline{b}_j.\sigma_i}$$

$$(\textsc{Cpl}\bigoplus\text{-L}):\ \frac{\Gamma,\ x{:}\bigoplus_{i\in I} \overline{a}_i.\rho_i \dashv^{\mathsf{cs}} \bigoplus_{j\in J} \overline{b}_j.\sigma_i \rhd^{\mathsf{inf}} f_i : \rho_i \dashv^{\mathsf{cs}} \bigoplus_{j\in J} \overline{b}_j.\sigma_i \quad (\forall i \in I)}{\Gamma \rhd^{\mathsf{inf}} \mathsf{rec}\, x.\bigvee_{i\in I}\langle a_i, \varepsilon\rangle.f_i : \bigoplus_{i\in I} \overline{a}_i.\rho_i \dashv^{\mathsf{cs}} \bigoplus_{j\in J} \overline{b}_j.\sigma_i}$$

$$(\textsc{Cpl}\bigoplus.\Sigma):\ \frac{\Gamma' \rhd^{\mathsf{inf}} f_i : \rho_i \dashv^{\mathsf{cs}} \sum_{j\in J} a_j.\sigma_j \quad (\forall i \in H) \qquad \Gamma' \rhd^{\mathsf{inf}} f_i : \rho_i \dashv^{\mathsf{cs}} \sigma_i \quad (\forall i \in K)}{\Gamma \rhd^{\mathsf{inf}} \mathsf{rec}\, x.(\bigvee_{h\in H}\langle a_h, \varepsilon\rangle.f_h) \vee (\bigvee_{k\in K}\langle a_k, \overline{a}_k\rangle.f_k) : \bigoplus_{i\in I} \overline{a}_i.\rho_i \dashv^{\mathsf{cs}} \sum_{j\in J} a_j.\sigma_j}\ (I = H \cup K, K \subseteq J)$$

$$\text{where}\quad \Gamma' = \Gamma,\ x{:}\bigoplus_{i\in I} \overline{a}_i.\rho_i \dashv^{\mathsf{cs}} \sum_{j\in J} a_j.\sigma_j.$$

$$(\textsc{Cpl}\Sigma.\bigoplus):\ \frac{\Gamma' \rhd^{\mathsf{inf}} f_j : \sum_{i\in I} a_i.\rho_i \dashv^{\mathsf{cs}} \sigma_j \quad (\forall j \in H) \qquad \Gamma' \rhd^{\mathsf{inf}} f_j : \rho_j \dashv^{\mathsf{cs}} \sigma_j \quad (\forall j \in K)}{\Gamma \rhd^{\mathsf{inf}} \mathsf{rec}\, x.(\bigvee_{h\in H}\langle \varepsilon, a_h\rangle.f_h) \vee (\bigvee_{k\in K}\langle \overline{a}_k, a_k\rangle.f_k) : \sum_{i\in I} a_i.\rho_i \dashv^{\mathsf{cs}} \bigoplus_{j\in J} \overline{a}_j.\sigma_j}\ (J = H \cup K, K \subseteq I)$$

$$\text{where}\quad \Gamma' = \Gamma,\ x{:}\sum_{i\in I} \overline{a}_i.\rho_i \dashv^{\mathsf{cs}} \bigoplus_{j\in J} a_j.\sigma_j$$

Figure 2: The inference system $\rhd^{\mathsf{inf}}$.

Theorem 3.3 (Soundness)**.** *If* $\Gamma \rhd^{\mathsf{inf}} f : \rho \dashv^{\mathsf{cs}} \sigma$ *then* $\Gamma \models f : \rho \dashv^{\mathsf{cs}} \sigma$.

Proof. (Sketch) It is possible to device a sound and complete system $\rhd$ for judgments of the shape $\Gamma \rhd f : \rho \dashv^{\mathsf{cs}} \sigma$, where f is a closed orchestrator and where Γ is a set of assumptions on closed orchestrators (not on variables as in $\rhd^{\mathsf{inf}}$). Now it can be proved that if f is closed and $\Gamma \rhd^{\mathsf{inf}} f : \rho \dashv^{\mathsf{cs}} \sigma$ is derivable, then for any θ such that $\theta \models \Gamma$ we have $\theta(\Gamma) \rhd f : \rho \dashv^{\mathsf{cs}} \sigma$, where $\theta(\Gamma)$ is the result of substituting all orchestrator variables x by $\theta(x)$. Then the thesis follows from the soundness of $\rhd$. □

The synthesis algorithm **Synth** is defined in Figure 3. Given a set of assumptions Γ, a client ρ and a server σ, the algorithm computes a set of orchestrators $\mathscr{F}$ such that for all $f \in \mathscr{F}$ a derivation of $\Gamma \rhd^{\mathsf{inf}} f : \rho \dashv^{\mathsf{cs}} \sigma$ exists. The algorithm essentially mimics the rules of the inference system of Figure 2. Intuitively, in case we are looking for orchestrators for $\rho = \bigoplus_{i\in I} \overline{a}_i.\rho_i$ and $\sigma = \bigoplus_{j\in J} \overline{a}_j.\sigma_i$ under the assumptions Γ, we notice that they can be inferred for such ρ and σ in system $\rhd^{\mathsf{inf}}$ only by means of rules $(\textsc{Cpl}\bigoplus\text{-R})$ or $(\textsc{Cpl}\bigoplus\text{-L})$ and that their form is, respectively, $\mathsf{rec}\, x.\bigvee_{i\in I}\langle a_i, \varepsilon\rangle.f_i$ or $\mathsf{rec}\, x.\bigvee_{j\in J}\langle \varepsilon, a_j\rangle.f_j$, where the f_is and the f_js are the orchestrators for the pairs ρ_i,σ and ρ,σ_j, respectively. This accounts for the fourth clause in the synthesis algorithm. We can prove the algorithm to be sound.

Lemma 3.4. *If* **Synth**$(\Gamma, \rho, \sigma) = \mathscr{F} \neq \emptyset$ *then, for all* $f \in \mathscr{F}$, $\Gamma \rhd^{\mathsf{inf}} f : \rho \dashv^{\mathsf{cs}} \sigma$ *is derivable.*

On the other hand, the algorithm is complete in the following sense:

Lemma 3.5. *If* $f : \rho \dashv\!|^{\mathsf{cs}} \sigma$ *and* **Synth**$(\emptyset, \rho, \sigma)$ *terminates then there exists* g *such that* $g \in$ **Synth**$(\emptyset, \rho, \sigma)$.

The g of the above lemma *represents* f. In particular it could be got by *delaying* the termination of the algorithm when the first clause is applicable. Moreover, it could be got out of f by replacing syncronous actions by pairs of asyncronous ones (or also by simply adding asyncronous actions). For instance, for

```
$\mathbf{Synth}(\Gamma,\rho,\sigma) =$
    if $x:\rho \dashv^{\mathsf{ds}} \sigma \in \Gamma$ then $\{x\}$
    else if $\rho = \mathbf{1}$ then $\{1\}$
    else if $\rho = \sum_{i\in I} a_i.\rho_i$ and $\sigma = \sum_{j\in J} a_j.\sigma_j$ then
        let $\Gamma' = \Gamma,\ x{:}\rho \dashv^{\mathsf{ds}} \sigma$ in
            $\bigcup_{i\in I}\{\,\mathsf{rec}\,x.\langle\overline{a}_i,\varepsilon\rangle.f \mid f \in \mathbf{Synth}(\Gamma',\rho_i,\sigma)\,\} \cup \bigcup_{j\in J}\{\,\mathsf{rec}\,x.\langle\varepsilon,\overline{a}_j\rangle.f \mid f \in \mathbf{Synth}(\Gamma',\rho,\sigma_j)\,\}$
    else if $\rho = \bigoplus_{i\in I} \overline{a}_i.\rho_i$ and $\sigma = \bigoplus_{j\in J} \overline{a}_j.\sigma_i$ then
        let $\Gamma' = \Gamma,\ x{:}\rho \dashv^{\mathsf{ds}} \sigma$ in
            $\{\,\mathsf{rec}\,x.\bigvee_{i\in I}\langle a_i,\varepsilon\rangle.f_i \mid f_i \in \mathbf{Synth}(\Gamma',\rho_i,\sigma)\,\} \cup \{\,\mathsf{rec}\,x.\bigvee_{j\in J}\langle\varepsilon,a_j\rangle.f_j \mid f_j \in \mathbf{Synth}(\Gamma',\rho,\sigma_j)\,\}$
    else if $\rho = \bigoplus_{i\in I} \overline{a}_i.\rho_i$ and $\sigma = \sum_{j\in J} a_j.\sigma_j$ then
        let $\Gamma' = \Gamma,\ x{:}\rho \dashv^{\mathsf{ds}} \sigma$ in
            $\{\,\mathsf{rec}\,x.(\bigvee_{h\in H}\langle a_h,\varepsilon\rangle.f_h) \vee (\bigvee_{k\in K}\langle a_k,\overline{a}_k\rangle.f_k)$
                $\mid I = H \cup K, K \subseteq J,\ f_h \in \mathbf{Synth}(\Gamma',\rho_h,\sigma),\ f_k \in \mathbf{Synth}(\Gamma',\rho_k,\sigma_k)\,\}$
            $\cup \bigcup_{j\in J}\{\,\mathsf{rec}\,x.\langle\varepsilon,\overline{a}_j\rangle.f \mid f \in \mathbf{Synth}(\Gamma',\rho,\sigma_j\,\}$
    else if $\rho = \sum_{i\in I} a_i.\rho_i$ and $\sigma = \bigoplus_{j\in J} \overline{a}_j.\sigma_j$ then
        let $\Gamma' = \Gamma,\ x{:}\rho \dashv^{\mathsf{ds}} \sigma$ in
            $\{\,\mathsf{rec}\,x.(\bigvee_{h\in H}\langle\varepsilon,a_h\rangle.f_h) \vee (\bigvee_{k\in K}\langle\overline{a}_k,a_k\rangle.f_k)$
                $\mid J = H \cup K, K \subseteq I,\ f_h \in \mathbf{Synth}((\Gamma',\rho,\sigma_h),\ f_k \in \mathbf{Synth}(\Gamma',\rho_k,\sigma_k)\,\}$
            $\cup \bigcup_{i\in I}\{\,\mathsf{rec}\,x.\langle\overline{a}_i,\varepsilon\rangle.f \mid f \in \mathbf{Synth}(\Gamma',\rho_i,\sigma)\,\}$
    else $\emptyset$
```

Figure 3: The algorithm **Synth**.

$\rho = \mathsf{rec}\,x.\overline{a}.x$ and $\sigma = \mathsf{rec}\,x.a.x$ the orchestrator $f = \langle a,\varepsilon\rangle.\langle\varepsilon,\overline{a}\rangle.\mathsf{rec}\,x.\langle a,\overline{a}\rangle.x$ correctly mediates between ρ and σ, the algorithm terminates, but $f \notin$ **Synth**$(\emptyset,\rho,\sigma)$. On the other hand, $g = \mathsf{rec}\,x.\langle a,\overline{a}\rangle.x$ belongs to **Synth**$(\emptyset,\rho,\sigma)$ and it is related to f in the sense above. It can be shown, besides, that if f is *respectful* in the sense of Sect. 4 below, there exists a respectful g in **Synth**$(\emptyset,\rho,\sigma)$.

It remains to show that **Synth** is terminating:

Lemma 3.6. *For all* Γ, ρ *and* σ, *the execution of* **Synth**(Γ,ρ,σ) *terminates.*

Proof. (Sketch) The proof is based on the fact that all session contracts in the recursive calls of **Synth** are a sub-expression of either ρ or σ or of a session contract in a judgment in Γ (which is finite). Since session contracts are regular trees, their sub-expressions are a finite set, so that the test $x{:}\rho \dashv^{\mathsf{ds}} \sigma \in \Gamma$ (where x is any variable) at the beginning of **Synth** cannot fail infinitely many times. □

Corollary 3.7. *The relation* $\dashv\!|^{\mathsf{ds}}$ *is decidable; moreover if* $\rho \dashv\!|^{\mathsf{ds}} \sigma$ *then it is possible to compute a set* $\mathscr{F}$ *of orchestrators for* ρ *and* σ

Recall that the computed orchestrators *represent* all the possible orchestrators, in the sense of the discussion after Lemma 3.5.

4 Respectfulness

The definition of orchestrators implies they have buffering capabilities. The sort of buffer taken into account in [19], as well as by us, is made of a number of bi-directional buffers (where only a finite subset is actually non empty), one for each possible name. A bi-directional buffer is actually made of two distinct buffers, one containing the messages received from the client that have to be delivered to the server, and the other one containing the messages received from the server that should be delivered to the client.

In [19] orchestrators are restricted to have bounded buffering capabilities and such a restriction is used in the proofs of several properties concerning contract orchestrators. In our setting we can eliminate that restriction, so allowing more client/server pairs to be compliant, like for instance $\mathtt{rec}\,x\,.\,a.x$ and $\mathtt{rec}\,x\,.\,\overline{b}.\overline{a}.x$, and the example in the introduction. We will now formalise the notion of buffer.

Definition 4.1 (Buffers)**.** *1. A bi-directional buffer $\mathbb{B}$ is a set of the form $\{\,{}^{c_a}a^{s_a} \mid a \in \mathscr{N}\,\}$ where, for any $a \in \mathscr{N}$, $c_a, s_a \in \mathbb{Z}$. The c_a in ${}^{c_a}a^{s_a}$ represents the number of a's in the part of the buffer containing messages sent by the client to the server. The s_a in ${}^{c_a}a^{s_a}$ represents the number of a's in the part of the buffer containing messages sent by the server to the client.*

2. *We define: $\tilde{\emptyset} = \{\,{}^{0}a^{0} \mid a \in \mathscr{N}\,\}$ and*

$$\begin{array}{rclcrcl}
\lfloor^{+}_{a}\mathbb{B} & = & (\mathbb{B} \setminus \{\,{}^{c_a}a^{s_a}\,\}) \cup \{\,{}^{c_a+1}a^{s_a}\,\} & \qquad & \mathbb{B}^{+}_{a}\rfloor & = & (\mathbb{B} \setminus \{\,{}^{c_a}a^{s_a}\,\}) \cup \{\,{}^{c_a}a^{s_a+1}\,\} \\
\lfloor^{-}_{a}\mathbb{B} & = & (\mathbb{B} \setminus \{\,{}^{c_a}a^{s_a}\,\}) \cup \{\,{}^{c_a-1}a^{s_a}\,\} & \qquad & \mathbb{B}^{-}_{a}\rfloor & = & (\mathbb{B} \setminus \{\,{}^{c_a}a^{s_a}\,\}) \cup \{\,{}^{c_a}a^{s_a-1}\,\}
\end{array}$$

3. *We denote by $|\mathbb{B}|_a$ the number of a's in the server-to-client part of the buffer, i.e. $|\mathbb{B}|_a = s_a$ and similarly for the client-to-server part, i.e. ${}_a|\mathbb{B}| = c_a$.*

4. *The state of a buffer $\mathbb{B}$ after an orchestration action μ will be denoted by $\mathbb{B}\mu$, defined by*

$$\begin{array}{rclcrclcrcl}
\mathbb{B}\langle \overline{a}, \varepsilon\rangle & = & \lfloor^{-}_{a}\mathbb{B} & & & & & & \mathbb{B}\langle \varepsilon, \overline{a}\rangle & = & \mathbb{B}^{-}_{a}\rfloor \\
 & & & \qquad & \mathbb{B}\langle \alpha, \overline{\alpha}\rangle & = & \mathbb{B} & \qquad & & & \\
\mathbb{B}\langle a, \varepsilon\rangle & = & \lfloor^{+}_{a}\mathbb{B} & & & & & & \mathbb{B}\langle \varepsilon, a\rangle & = & \mathbb{B}^{+}_{a}\rfloor
\end{array}$$

5. *By $\mathbb{B}\boldsymbol{\mu}$ we denote the buffer $\mathbb{B}$ after the sequence $\boldsymbol{\mu}$ of orchestration actions.*

In Definition 2.8 we considered the relation $\dashv\!\mid^{\mathsf{cs}}$, which we have studied so far. This is however much weaker than expected, and it is time to face the issue. Consider the simple orchestrated system

$$\overline{a}.\overline{b} \parallel_f a.c.d \quad \text{where } f = \langle a, \overline{a}\rangle.\langle b, \varepsilon\rangle.\mathtt{1}.$$

It is easy to check that $f : \overline{a}.\overline{b} \dashv\!\mid^{\mathsf{cs}} a.c.d$ since f is strict for the given client/server pair and $\overline{a}.\overline{b} \parallel_f a.c.d \overset{\boldsymbol{\mu}}{\Longrightarrow} \mathbf{1} \parallel_{\mathtt{1}} c.d \not\longrightarrow$, where $\boldsymbol{\mu} = \langle a, \overline{a}\rangle\langle b, \varepsilon\rangle$. It is definitely true that all the client's "requests" have been satisfied, but not all by the server! The action $\overline{b}$ of the client has been taken care of exclusively by the orchestrator, which in that case has not acted simply as a mediator, but has effectively participated to the completion of the client's requests.

So, in order to strengthen Definition 2.8 (i), in case $\rho' \parallel_{f'} \sigma' \not\longrightarrow$, we have to impose some conditions on the client-to-server buffer associated to f'; in particular, that it should be empty. Of course, a similar condition must hold also for infinite interactions; this implies that in an infinite interaction, for any possible name, say a, used by the orchestrator, the latter cannot indefinitely perform input actions for a from the client (even if interspersed with actions for other names) without ever delivering an a to the server. We must therefore forbid a client like $\mathtt{rec}\,x\,.\,\overline{a}.\overline{c}.x$ to be compliant with the server $\mathtt{rec}\,x\,.\,c.x$ by means of the orchestrator $\mathtt{rec}\,x\,.\langle a, \varepsilon\rangle.\langle c, \overline{c}\rangle.x$. Orchestrated finite and infinite interaction sequences which do not correspond to unwanted situations like those just sketched will be called **client-respectful**.

Even if the notion of compliance enforces the sense of the bias towards the client (any client request must be eventually satisfied by the server), some conditions need to be imposed on the part of interactions on behalf of the server. In fact, we wish to prevent a server to be compliant with a client by means of an orchestrator that, from a certain moment on, interacts infinitely many times with the server only, like in the orchestrated system

$$\overline{a}.b \,\|_f\, \mathsf{rec}\,x.\overline{c}.\overline{b}.x \quad \text{where } f = \langle a,\varepsilon\rangle.\mathsf{rec}\,x.\langle \varepsilon,c\rangle.\langle \varepsilon,b\rangle.x$$

We wish to prevent this kind of infinite interaction that we dub **definitely server-inputted**. Notice that, however, we can permit interactions in which the orchestrator can perform the input of some a from the server infinitely many times, without ever performing an output of a to the client, like it happens for `wind` in the example in the introduction.

We observe that the problem – whether an orchestrator will ever engage in any of the aforementioned pathological interactions – might well be undecidable for contracts in general; indeed, it shares similarities with, for example, termination of two-counter machines [17]. However, we stress that we are in the restricted setting of session contracts, which suffices to make such properties decidable.

Among the properties we have to take care of, one is that in an interaction sequence there cannot exist an orchestrator action removing an element from an empty buffer, i.e. a **sound** sequence never sends an element a to a server or to a client if the a has not been previously received. We call the sum of all the above properties **respectfulness**.

Definition 4.2. *Given $\boldsymbol{\mu} \in \mathbf{OrchAct}^* \cup \mathbf{OrchAct}^\infty$, we define ${}_a{\downharpoonleft}\boldsymbol{\mu}$, its left-restriction to a name a, as follows (λ is the empty sequence):*

$$\begin{array}{lcll}
{}_a{\downharpoonleft}\lambda & = & \lambda, & \\
{}_a{\downharpoonleft}(\mu\boldsymbol{\mu}') & = & \mu\, {}_a{\downharpoonleft}\boldsymbol{\mu}', & \textit{if } \mu \in \{\, \langle \varepsilon,\overline{a}\rangle, \langle a,\varepsilon\rangle \,\}, \\
{}_a{\downharpoonleft}(\mu\boldsymbol{\mu}') & = & {}_a{\downharpoonleft}\boldsymbol{\mu}' & \textit{otherwise.}
\end{array}$$

Definition 4.3 (Respectful sequences and orchestrators)**.** *Let $\boldsymbol{\mu} \in \mathbf{OrchAct}^* \cup \mathbf{OrchAct}^\infty$ and $\mu \in \mathbf{OrchAct}$.*

a) Given $S \subseteq \mathbf{OrchAct}$, we say $\boldsymbol{\mu}$ to be definitely-*S whenever:*

$$\exists k\; \forall m \geq k\; [\textit{the } m\textit{-th element of } \boldsymbol{\mu} \textit{ belongs to } S];$$

For sets that are singletons we write ‘definitely-μ’ instead of ‘definitely-$\{\,\mu\,\}$.’

b) We say $\boldsymbol{\mu}$ to be a sound sequence *whenever:*

$$\forall a \in \mathcal{N}\; \forall n \leq |\boldsymbol{\mu}|\; [\, {}_a|\tilde{\emptyset}\mu_1 \cdots \mu_n\,| \geq 0 \;\; \textit{and} \;\; |\tilde{\emptyset}\mu_1 \cdots \mu_n\,|_a \geq 0]$$

c) We say $\boldsymbol{\mu}$ to be client-respectful *sequence whenever, for any $a \in \mathcal{N}$:*

$${}_a{\downharpoonleft}\boldsymbol{\mu} \textit{ is finite and } {}_a|\tilde{\emptyset}\boldsymbol{\mu}\,| = 0 \quad \textit{or} \quad {}_a{\downharpoonleft}\boldsymbol{\mu} \textit{ is infinite and non-definitely-}\langle a,\varepsilon\rangle$$

d) We say $\boldsymbol{\mu}$ to be non-definitely server–inputted *whenever:*

$$\boldsymbol{\mu} \textit{ is infinite} \quad \Longrightarrow \quad \boldsymbol{\mu} \textit{ is non-definitely-}\{\, \langle \varepsilon,a\rangle \mid a \in \mathcal{N} \,\}$$

e) We say $\boldsymbol{\mu}$ to be respectful *whenever $\boldsymbol{\mu}$ is sound, client-respectful and non-definitely server-inputted.*

f) We say that an orchestrator f is respectful *whenever every $\boldsymbol{\mu} \in \mathsf{MaxTr}(f)$ is so.*

We will look now at a few examples in order to get a better intuition about the above definition.

Example 4.4.

- *The finite sequence $\langle a,\varepsilon\rangle.\langle \varepsilon,\overline{b}\rangle.\langle \varepsilon,\overline{a}\rangle$ is not respectful since it is not sound. In fact, for the name b, we have that $|\tilde{\emptyset}.\langle a,\varepsilon\rangle.\langle \overline{b},\varepsilon\rangle\,|_b = -1 < 0$.*
- *The sequence $\langle a,\varepsilon\rangle.\langle b,\varepsilon\rangle.\langle \varepsilon,\overline{a}\rangle$ instead, is sound, but nonetheless it is not client-respectful, since it is not infinite and for the name b we have ${}_b|\tilde{\emptyset}\langle a,\varepsilon\rangle.\langle b,\varepsilon\rangle.\langle \varepsilon,\overline{a}\rangle\,| = 1 \neq 0$.*
- *The orchestrator $f = \langle c,\overline{c}\rangle.\mathsf{rec}\,x.(\langle \overline{a},a\rangle \vee \langle c,\varepsilon\rangle.\langle b,\overline{b}\rangle.x)$ is not respectful since it is not client-respectful. In fact, for the sequence $\boldsymbol{\mu} = \langle c,\overline{c}\rangle.\langle c,\varepsilon\rangle.\langle b,\overline{b}\rangle.\langle c,\varepsilon\rangle.\langle b,\overline{b}\rangle \cdots \in \mathsf{MaxTr}(f)$ and the name c, we have that ${}_c{\downharpoonleft}\boldsymbol{\mu}$ is infinite and ${}_c{\downharpoonleft}\boldsymbol{\mu} = \langle c,\varepsilon\rangle.\langle c,\varepsilon\rangle.\langle c,\varepsilon\rangle \cdots$ is definitely-$\langle c,\varepsilon\rangle$. In fact, from the very first element on it is made of $\langle c,\varepsilon\rangle$ actions.*

- *The orchestrator* $f = \langle c,\overline{c}\rangle.\mathsf{rec}\,x.(\langle \overline{a},a\rangle \vee \langle \varepsilon,b\rangle.\langle \varepsilon,c\rangle.x)$ *is not respectful since it is not definitely server-inputted. In fact, the infinite sequence* $\boldsymbol{\mu} = \langle c,\overline{c}\rangle.\langle \varepsilon,b\rangle.\langle \varepsilon,c\rangle.\langle \varepsilon,b\rangle.\langle \varepsilon,c\rangle \cdots \in \mathsf{MaxTr}(f)$ *is definitely-*$\{\,\langle \varepsilon,a\rangle \mid a \in \mathscr{N}\,\}$*. The orchestrator* `f` *in the introduction, instead, is non-definitely server-inputted, and also respectful, as a matter of fact.*

Remark 4.5. *By Definition 4.2, the sequence* ${}_{a}\!\downharpoonleft\!\boldsymbol{\mu}$ *in Definition 4.3(c) cannot contain synchronous orchestration actions like* $\langle a,\overline{a}\rangle$*. Hence, for example, the orchestrator* $g = \langle a,\varepsilon\rangle.\mathsf{rec}\,x.\langle a,\overline{a}\rangle.x$ *is not client-respectful, and so it is not respectful at all. This is because the first a coming from the client will never be delivered to the server since any subsequent output* $\overline{a}$ *will be paired with a further input of a. This might be irrelevant when distinct occurrences of the same message are indistinguishable, but in general the number of input-output actions matters.*

On the other hand forcing the orchestrator to immediately forward a message is a desirable capability, which would be definitely lost by equating $\langle a,\varepsilon\rangle.\langle \varepsilon,\overline{a}\rangle$ *to* $\langle a,\overline{a}\rangle$*, and by ruling out the latter.*

We can now properly define the full notion of compliance and characterise it.

Definition 4.6 (Orchestrated Session Compliance)**.** *i) We say that a client* ρ *is compliant with a server* σ *through the orchestration of* f*, and denote this by* $f : \rho \dashv\!\mid \sigma$*, whenever*

a) $\rho \parallel_f \sigma \overset{\boldsymbol{\mu}}{\Longrightarrow} \rho' \parallel_{f'} \sigma' \not\longrightarrow$ *implies* $\rho' = \mathbf{1}$ *and* $\boldsymbol{\mu}$ *is respectful, and*

b) $\rho \parallel_f \sigma \overset{\boldsymbol{\mu}}{\Longrightarrow}$ *with* $\boldsymbol{\mu} \in \mathbf{OrchAct}^{\infty}$ *implies* $\boldsymbol{\mu}$ *is respectful.*

ii) We write $\rho \dashv\!\mid \sigma$ *whenever there exists an orchestrator* f *such that* $f : \rho \dashv\!\mid \sigma$*.*

Notice that we cannot define orchestrated compliance by simply imposing f to be respectful in Def. 4.6(i), since that would prevent the possibility of $\overline{a}$ be compliant with a through the mediation of the orchestrator $\langle a,\overline{a}\rangle \vee \langle \varepsilon,\overline{b}\rangle$. This orchestrator is not respectful, but its sequences of actions in any possible orchestration between $\overline{a}$ and a are respectful.

We can show that, if compliance could be obtained by means of a non-respectful orchestrator, it is always possible to get it through a respectful one. Besides, we can show the correspondence between $\dashv\!\mid$ and $\dashv\!\mid^{\mathsf{cfs}}$.

Proposition 4.7. *i)* $f : \rho \dashv\!\mid \sigma \implies \exists f'\,[\,f' : \rho \dashv\!\mid \sigma$ *such that* f' *is* ρ*-*σ *strict*$\,]$.

ii) $f : \rho \dashv\!\mid \sigma$ *and* f *is* ρ*-*σ *strict* $\Leftrightarrow$ $f : \rho \dashv\!\mid^{\mathsf{cfs}} \sigma$ *and* f *is respectful.*

iii) $\rho \dashv\!\mid \sigma \Leftrightarrow \exists f\,[\,f : \rho \dashv\!\mid^{\mathsf{cfs}} \sigma$ *where* f *is respectful*$\,]$.

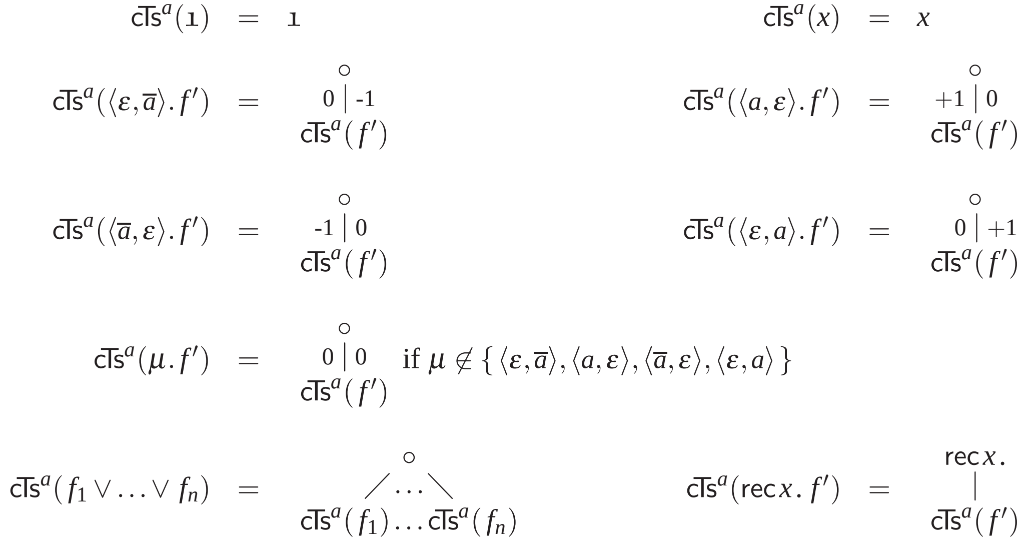
In order to show decidability, we provide a characterisation of respectfulness based on the notion of buffer-aware trees and its related labelings below.

Definition 4.8 (Buffer-aware trees of f)**.** *a) Let* $a \in \mathscr{N}$*. We define the* buffer-aware a-tree *of an orchestrator* f*, denoted by* $\mathsf{cTs}^{a}(f)$*, as the tree defined by induction in Figure 4. The edges of the tree have a left- and a right-weight denoting, respectively, the increment of the client-to-server and of the server-to-client buffer for the name 'a' caused by the orchestration actions performed by* f*.*

Given an edge e of a buffer-aware a-tree t, we denote is left (resp. right) weight by $\mathsf{lw}^{t}(e)$ *(resp.* $\mathsf{rw}^{t}(e)$*).*

b) We define the buffer-aware $*$-tree *of an orchestrator* f*, denoted by* $\mathsf{cTs}^{*}(f)$*, as the tree with the same nodes and edges as any* $\mathsf{cTs}^{a}(f)$*, but such that the left (resp. right) weight of an edge e is* $\sum_{a\in\mathscr{N}} \mathsf{lw}^{\mathsf{cTs}^{a}(f)}(e)$ *(resp.* $\sum_{a\in\mathscr{N}} \mathsf{rw}^{\mathsf{cTs}^{a}(f)}(e)$ *).*

Note that the left and right weights of the edges of a buffer-aware $*$-tree of an orchestrator f are either 0, -1, or $+1$.



$\mathsf{cTs}^a(\mathtt{1}) = \mathtt{1}$
$\mathsf{cTs}^a(x) = x$
$\mathsf{cTs}^a(\langle\varepsilon,\overline{a}\rangle.f') =$ ○ 0 | -1 $\mathsf{cTs}^a(f')$
$\mathsf{cTs}^a(\langle a,\varepsilon\rangle.f') =$ ○ +1 | 0 $\mathsf{cTs}^a(f')$
$\mathsf{cTs}^a(\langle\overline{a},\varepsilon\rangle.f') =$ ○ -1 | 0 $\mathsf{cTs}^a(f')$
$\mathsf{cTs}^a(\langle\varepsilon,a\rangle.f') =$ ○ 0 | +1 $\mathsf{cTs}^a(f')$
$\mathsf{cTs}^a(\mu.f') =$ ○ 0 | 0 $\mathsf{cTs}^a(f')$ if $\mu \notin \{\langle\varepsilon,\overline{a}\rangle, \langle a,\varepsilon\rangle, \langle\overline{a},\varepsilon\rangle, \langle\varepsilon,a\rangle\}$
$\mathsf{cTs}^a(f_1 \vee \ldots \vee f_n) =$ ○ … $\mathsf{cTs}^a(f_1)\ldots\mathsf{cTs}^a(f_n)$
$\mathsf{cTs}^a(\mathsf{rec}\,x.\,f') =$ $\mathsf{rec}\,x.$ $\mathsf{cTs}^a(f')$

Figure 4: Buffer-aware *a*-tree

Definition 4.9 (Buffer-labelling of $\mathsf{cTs}^a(f)$)**.** *We define the* buffer-labelling of $\mathsf{cTs}^a(f)$ *by labelling its nodes with left and right labels as follows: given a node N and the path P in* $\mathsf{cTs}^a(f)$ *from the root to N, we left-label N with the sum of all the left-weights of the edges in P, whereas we right-label N with the sum of all the right-weights of the edges in P.*

We now provide characterisations for the properties defining respectfulness.

Definition 4.10 (Sound buffer-labelling)**.** *The buffer-labelling of* $\mathsf{cTs}^a(f)$ *is* sound *whenever*

a) there is no negative left-label and no negative right-label and

b) for any leaf x and corresponding $\mathsf{rec}\,x.$ *node, if k is the left (resp. right) label of x and h is the left (resp. right) label of* $\mathsf{rec}\,x.$*, then:* $k - h \geq 0$.

Proposition 4.11. *f is sound* $\Leftrightarrow$ *for any $a \in \mathcal{N}$, the buffer-labelling of* $\mathsf{cTs}^a(f)$ *is sound.*

Proof. ($\Leftarrow$) By the labelling, it is impossible to get a non-client-respectful sequence out of f.
($\Rightarrow$) By contraposition; assume that for a name $b \in \mathcal{N}$, the buffer-labelling of $\mathsf{cTs}^b(f)$ be unsound. Then we have two cases to consider:

(a) There is a negative label. We then get immediately an unsound sequence.

(b) There exists a leaf x and its corresponding $\mathsf{rec}\,x.$ node, where k is the left(or right-)-label of x and h is the left-(or right-)label of $\mathsf{rec}\,x.$, s.t. $k - h < 0$. It is immediate to get an unsound sequence. □

We say that a node *gets to* $\mathtt{1}$ whenever its subtree contains a $\mathtt{1}$ node.

Definition 4.12 (Client-respectful buffer-labelling)**.** *The buffer-labelling of* $\mathsf{cTs}^a(f)$ *is* client-respectful *whenever*

a) any $\mathtt{1}$ *node is left-labelled with* 0*;*

b) for any leaf x and corresponding node of its binder $\mathsf{rec}\,x.$*, if k is the left-label of x and h is the left-label of* $\mathsf{rec}\,x.$*, then*

1) *if the* $\mathtt{rec}\,x.$ *node gets to* $\mathtt{1}$*, then* $h = k$*;*
2) *otherwise, if all the left-labels of the edges from* x *to* $\mathtt{rec}\,x.$ *are* 0 *then* $h = 0$*;*

c) *for any path from a leaf* x *to its corresponding* $\mathtt{rec}\,x.$ *node, either no edge is right-weighted with* $+1$ *or there is at least an edge with right-weight* -1.

Proposition 4.13.
f *is client-respectful* $\quad\Leftrightarrow\quad$ *for any* $a \in \mathscr{N}$*, the buffer-labelling of* $\mathsf{cTs}^{a}(f)$ *is client-respectful.*

Proof. ($\Leftarrow$) By the labeling rule it is impossible to get a non client-respectful sequence out of f. For finite sequences this impossibility is guaranteed by clauses (a) and (b) of Definition 4.12, for infinite ones by clause (c).
($\Rightarrow$) By contraposition; assume that for a name $b \in \mathscr{N}$, the buffer-labelling of $\mathsf{cTs}^{b}(f)$ be non-client-respectful. We consider the four possible cases:

1. A label of a $\mathtt{1}$ leaf is not 0. In that case we immediately get a finite sequence out of f which is non-client-respectful.
2. There is a node x labelled with k and its corresponding node $\mathtt{rec}\,x.$ gets to $\mathtt{1}$ and it is labelled with h, with $k \neq h$. Then the sequence out of f corresponding to going to $\mathtt{rec}\,x.$, then from node $\mathtt{rec}\,x.$ to x a non negative number of times n and finally to the $\mathtt{1}$ node cannot be client-respectful, since at the end the client-to-server buffer for b would have $n * (h-k)$ elements in it.
3. There is a node x labelled with k, its corresponding node $\mathtt{rec}\,x.$ does not get to $\mathtt{1}$, all the left-labels of the edges from x to $\mathtt{rec}\,x.$ are 0 and $h = 0$. In that case the trace $\boldsymbol{\mu}$ corresponding to the infinite path starting from the root and then keeping indefinitely on passing through $\mathtt{rec}\,x.$ and x is such that ${}_{b}\rfloor\boldsymbol{\mu}$ is finite and $|\tilde{\emptyset}({}_{b}\rfloor\boldsymbol{\mu})| \neq 0$.
4. there exists a path from a leaf x to its corresponding $\mathtt{rec}\,x.$ such that there are some right-weighted edges right-weighted with $+1$ and no edge with right-weight -1. Then it is immediate to get an infinite definitely server-inputted sequence out of f which is definitely-$\langle b, \varepsilon\rangle$ and hence not client-respectful. □

Definition 4.14 (Non definitely server-inputted $*$-tree)**.** *Given an orchestrator* f*, its* $*$*-tree* $\mathsf{cTs}^{*}(f)$ *is* non-definitely server-inputted *whenever, for any path from a leaf* x *to its corresponding* $\mathtt{rec}\,x.$ *node, either no edge is right-weighted with* $+1$ *or there is at least an edge with right-weight* -1.

Proposition 4.15. $\quad f$ *is not definitely server-inputted* $\quad\Leftrightarrow\quad$ $\mathsf{cTs}^{*}(f)$ *is non-definitely server-inputted.*

Proof. ($\Leftarrow$) By the labelling it is impossible to get a definitely server-inputted sequence out of f.
($\Rightarrow$) By contraposition; assume that $\mathsf{cTs}^{*}(f)$ be definitely server-inputted. So there exists a path from a leaf x to its corresponding $\mathtt{rec}\,x.$ such that there are some right-weighted edges right-weighted with $+1$ and no edge with right-weight -1. Then it is immediate to get a definitely server-inputted sequence out of f. □

Theorem 4.16. *Orchestrator respectfulness is decidable.*

From the above result and from decidability of $\dashv\!\mid^{\mathsf{cs}}$ (Corollary 3.7) we can get decidability of $\dashv\!\mid$. The algorithm to decide whether $\rho \dashv\!\mid \sigma$ will first compute $\mathscr{F} = \textbf{Synth}(\emptyset, \rho, \sigma)$; then if $\mathscr{F} \neq \emptyset$ it suffices to check whether there is a strict and respectful $f \in \mathscr{F}$, which is a decidable problem by the above.

Theorem 4.17. *Given* ρ *and* σ*, it is decidable whether* $\rho \dashv\!\mid \sigma$*.*

We conclude by observing that in [19] the lack of unbounded buffering capabilities prevents orchestrators to be used to ensure client compliance with a server that might send an unbounded number of unnecessary outputs. To let such sort of interaction possible, in [2] the notion of `skp`-compliance (dubbed $\dashv^{\mathtt{skp}}$) was investigated for session contracts, where a client is compliant with a server whenever all its requests can be satisfied thanks to the possibility of discarding a (possibly unbounded) number of unnecessary server outputs. Interactions of this sort can actually be carried out by means of our session orchestrators, since it is possible to prove that $\rho \dashv^{\mathtt{skp}} \sigma$ implies $\rho \dashv\!\mid^{\mathtt{cs}} \sigma$. In the example in the introduction, in fact, the `wind` information is unbounded and "discarded" by the orchestrator.

5 Related and future work

The notion of compliance naturally induces a substitutability relation on servers that may be used for implementing contract-based query engines (see [19] for a detailed discussion). Hence it seems worthwhile to investigate the session sub-contract relation induced by our orchestrated compliance on session contracts. Whereas server substitutability is at the core of the results in [19], we deem it relevant to investigate also *client substitutability*, in the style of what was done in [1, 3] for session contract and in [10] for the more general notion of contract.

An approach to the formal description of service contracts in terms of automata has been recently developed in [6]. The notion of *contract automaton* is related to that of *contract* as well as of *session contract*. Besides, the notion of *contract agreement* in [6] somewhat resembles that of *compliance*. In the framework of that paper, orchestrators are synthesised to enforce contract composition to adhere to the requirements for contract agreement. Even if the authors of [6] work on the overall satisfaction in a multiparty composition of principals, it is definitely worthwhile, as a future investigation, to study the relation between the notion of orchestration, as developed in [19] and in the present paper, and the approach of [6], which in turn has been related in [7] to the model of choreography of communicating finite state machines (CFMS) [11]. For what concerns *session contracts* in particular, the investigation of the correspondence with the above mentioned formalisms could start from the result concerning the correspondence of *binary* session types with a particular two-communicating-machines subclass (see [15] for references). Such a correspondence between session types and communicating machines has been pushed further to the multiparty setting in [15].

Many properties of the model of CFSM which are untractable ceases to be so when Bags, instead of - or together with - FIFO queues are taken into account [14]. The similarity of contracts and session contracts with the CFSM model suggests to investigate the use of bags for session-contract interactions to reduce decidability problems in our context to problems in the CFSM model with bags. What does a bag correspond to in our context is however not immediate to device. In fact, by putting a bag in between $\overline{a}.\overline{b}$ and $a+b$ would result in a number of possible non-deterministic evolutions of the system: as soon as a is in the bag, it could be used as input for the server; or, in case both a and b get into the bag, the server could non-deterministically choose amongst them; etc. Such a behaviour of the system, however, goes far beyond the session setting we are in, where non-determinism is restricted to occur only inside the client and server.

Session contracts have been also investigated in papers like [4, 5] where, overloading the name, they also have been dubbed *session types*. In [4] the authors establish a relation between session contracts and a model based on game-theoretic notions, showing that compliance corresponds to the existence of particular winning strategies. It should be interesting to investigate the meaning and role of the notion of orchestration in such a game-theoretical setting.

Acknowledgments. We are grateful to the referees for their helpful and meaningful advices. The interaction with them has been pleasing and fruitful thanks to the forum tool provided by the workshop organisation. We also wish to thank Mariangiola Dezani for her everlasting support.

References

[1] Franco Barbanera & Ugo de'Liguoro (2010): *Two notions of sub-behaviour for session-based client/server systems*. In: *PPDP*, ACM Press, pp. 155–164, doi:10.1145/1836089.1836109.

[2] Franco Barbanera & Ugo de' Liguoro (2014): *Loosening the notions of compliance and sub-behaviour in client/server systems*. In: Proceedings 7th *ICE 2014*, *EPTCS* 166, pp. 94–110, doi:10.4204/EPTCS.166.10.

[3] Franco Barbanera & Ugo de' Liguoro (2014): *Sub-behaviour relations for session-based client/server systems*. *Math. Struct. in Comp. Science*, doi:10.1017/S096012951400005X. To appear, published online.

[4] Massimo Bartoletti, Tiziana Cimoli & G. Michele Pinna (2014): *A note on two notions of compliance*. In: Proceedings 7th *ICE 2014*, *EPTCS* 166, pp. 86–93, doi:10.4204/EPTCS.166.9.

[5] Massimo Bartoletti, Alceste Scalas & Roberto Zunino (2014): *A semantic deconstruction of session types*. In: *Proc. CONCUR*, pp. 402–418, doi:10.1007/978-3-662-44584-6$_$28.

[6] Davide Basile, Pierpaolo Degano & Gian Luigi Ferrari (2014): *Automata for Analysing Service Contracts*. In: *TGC 2014*, *LNCS* 8902, pp. 34–50, doi:10.1007/978-3-662-45917-1-3.

[7] Davide Basile, Pierpaolo Degano, Gian-Luigi Ferrari & Emilio Tuosto (2014): *From Orchestration to Choreography through Contract Automata*. In: Proc. *ICE'14*, *EPTCS* 166, pp. 67–85, doi:10.4204/EPTCS.166.8.

[8] Giovanni Bernardi & Matthew Hennessy (2012): *Modelling session types using contracts*. In: *Proceedings of 27th Annual ACM SAC '12*, ACM, New York, NY, USA, pp. 1941–1946, doi:10.1145/2231936.2232097.

[9] Giovanni Bernardi & Matthew Hennessy (2014): *Modelling session types using contracts*. *Math. Struct. in Comp. Science*, doi:10.1017/S0960129514000243. To appear, published online.

[10] Giovanni Bernardi & Matthew Hennessy (2015): *Mutually Testing Processes*. *24h CoRR* abs/1502.06360, doi:10.1007/978-3-642-40184-8$_$6.

[11] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. *JACM* 30(2), pp. 323–342, doi:10.1145/322374.322380.

[12] S. Carpineti, G. Castagna, C. Laneve & L. Padovani (2006): *A formal account of contracts for Web Services*. In: *WS-FM*, *LNCS* 4184, Springer, pp. 148–162, doi:10.1007/11841197_10.

[13] Giuseppe Castagna, Nils Gesbert & Luca Padovani (2009): *A theory of contracts for Web services*. *ACM Trans. on Prog. Lang. and Sys.* 31(5), pp. 19:1–19:61, doi:10.1145/1538917.1538920.

[14] L. Clemente, F. Herbreteau & G. Sutre (2014): *Decidable Topologies for Communicating Automata with FIFO and Bag Channels*. In: *Proc. CONCUR'14*, *LNCS* 8704, doi:10.1007/978-3-662-44584-6$_$20.

[15] Pierre-Malo Deniélou & Nobuko Yoshida (2012): *Multiparty Session Types Meet Communicating Automata*. In: *ESOP*, pp. 194–213, doi:10.1007/978-3-642-28869-2$_$10.

[16] Kohei Honda, Vasco T. Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Disciplines for Structured Communication-based Programming*. In: *ESOP*, *LNCS* 1381, Springer, pp. 22–138, doi:10.1007/BFb0053567.

[17] O. H. Ibarra, J. Su, Z. Dang, T. Bultan & R. Kemmerer (2000): *Counter Machines: Decidable Properties and Applications to Verification Problems*. In: *MFCS 2000*, *LNCS* 1893, doi:10.1007/3-540-44612-5-38.

[18] Cosimo Laneve & Luca Padovani (2007): *The Must Preorder Revisited: An Algebraic Theory for Web Services Contracts*. In: *CONCUR'07*, *LNCS* 4703, Springer, pp. 212–225, doi:10.1007/978-3-540-74407-8$_$15.

[19] Luca Padovani (2010): *Contract-Based Discovery of Web Services Modulo Simple Orchestrators*. *Theoretical Computer Science* 411, pp. 3328–3347, doi:`10.1016/j.tcs.2010.05.002`.

[20] Chris Peltz (2003): *Web Services Orchestration and Choreography*. *Computer* 36(10), pp. 46–52, doi:`10.1109/MC.2003.1236471`.