The pairing of contracts and session types

Cosimo Laneve[1] and Luca Padovani[2]

[1] Department of Computer Science, University of Bologna
[2] Information Science and Technology Institute, University of Urbino

Dedicated to Ugo Montanari in occasion of his 65th birthday

Abstract. We pair session types and contracts using two encodings. The encoding of session types accommodates width and depth subtyping, two properties that partially hold in contracts. The encoding of contracts accommodates complex synchronization patterns, since session types own a simple control protocol. The encodings allow one to use the two formalisms interchangeably, within the context of dyadic interactions.

1 Introduction

Service Oriented technologies and Web Services have been recently proposed as a new way of distributing and organizing complex applications across the Internet. The success of these technologies has fostered the development of formal methods for statically analyzing and verifying the behavior of concurrent and distributed systems. Two such methods – session types [13, 10, 9] and contracts [3, 14, 4] – aim at describing the communication protocol implemented by services. These methods provide a foundation for statically checking that a process implements a given communication protocol and formally characterize *compliance* (when a client interacts successfully with a service) and *safe replacement* (when it is possible to replace a process with another one).

Session types and contracts find their origins in two different domains: the former ones derive from the domain of type theory and type systems, whereas contracts are more related to the study of behavioral equivalences, such as bisimulation and testing equivalence [6, 12]. Both languages are equipped with similar constructors. For example, the session type

$$S \overset{\text{def}}{=} \mu x.\&\langle \texttt{Login} : \oplus\langle \texttt{Wrong} : x; \texttt{Ok} : \&\langle \texttt{VoteA} : \texttt{end}; \texttt{VoteB} : \texttt{end}\rangle; \texttt{Cheat} : \texttt{end}\rangle\rangle$$

and the contract

$$\mathrm{I}[\sigma] \overset{\text{def}}{=} \mathrm{I}[\texttt{rec}\ x.\texttt{Login}.(\overline{\texttt{Wrong}}.x \oplus \overline{\texttt{Ok}}.(\texttt{VoteA} + \texttt{VoteB}) \oplus \overline{\texttt{Cheat}})]$$

where $\mathrm{I} = \{\texttt{Login}, \overline{\texttt{Wrong}}, \overline{\texttt{Ok}}, \overline{\texttt{Cheat}}, \texttt{VoteA}, \texttt{VoteB}\}$, represent a simple service for an online ballot between two candidates `A` and `B`. Before a client is allowed to vote, he must provide a valid login token that the system uses for ensuring that preferences are expressed at most once, for otherwise the voter is identified as a cheater.

In contracts, the actions such as `Login` and $\overline{\texttt{Cheat}}$ represent atomic communications between the voter and the service and we have two binary operators for encoding alternatives: a $+$ indicates an *external choice* (the voter chooses the candidate) whereas a $\oplus$ indicates an *internal choice* (the service decides whether the login is valid or not). In session types we have *branches* $\&\langle\cdots\rangle$ and *choices* $\oplus\langle\cdots\rangle$ that play the same roles played by $+$ and $\oplus$. However, branches indicate the receipt of a label expressing the decision of the voter, whereas choices indicate that the service emits a label expressing the alternative it has chosen. Hence, the direction of the exchanged messages is encoded in the type of alternative, rather than in the labels themselves. Also, while actions in contracts may encode the exchange of data between the voter and the service, labels in session types are only meant to implement the control part of the protocol. In fact, typical presentations of session types admit additional constructs for representing the exchange of data, which we omit in this paper as they are irrelevant for the results that follow.

The relationship between session types and contracts also regards the semantics, although, in this case, their different origin is evident because the relations are mostly the opposite. For example, the session type

$$T \stackrel{\text{def}}{=} \&\langle \mathtt{Login} : \oplus\langle \mathtt{Ok} : \&\langle \mathtt{VoteA} : \mathtt{end}; \mathtt{VoteB} : \mathtt{end}\rangle\rangle\rangle$$

and the contract

$$\mathrm{I}[\tau] \stackrel{\text{def}}{=} \mathrm{I}[\mathtt{Login}.\overline{\mathtt{Ok}}.(\mathtt{VoteA} + \mathtt{VoteB})]$$

both describe a ballot service that does not care about cheaters and always accepts login tokens regardless their validity. It turns out that T is a *subtype* S (notation $T \leq S$) and $\mathrm{I}[\sigma]$ is a *subcontract* of $\mathrm{I}[\tau]$ (notation $\mathrm{I}[\sigma] \preceq \mathrm{I}[\tau]$). That is, the subtype relation $T \leq S$ embodies the notion of safe substitutability: every term having type S may be replaced by a term having type T without affecting the context in a sensible way. The subcontract relation $\mathrm{I}[\sigma] \preceq \mathrm{I}[\tau]$ embodies the notion of successful interaction (called *compliance*): the set of clients succeeding in interacting with a service of contract $\mathrm{I}[\sigma]$ also succeed with a service of contract $\mathrm{I}[\tau]$.

There are also some differences between the theory of session types and the one of contracts. In one direction, these differences mainly regard the so-called *width* and *depth subtyping*. Session types, much alike object-oriented type disciplines, enjoy the property $\&\langle \mathtt{VoteA} : \mathtt{end}; \mathtt{VoteB} : \mathtt{end}\rangle \leq \&\langle \mathtt{VoteA} : \mathtt{end}\rangle$ (the ballot service can be extended with more candidates without invalidating former voters), namely width extensions of capabilities is always possible. In contracts this is not the case. To discuss the point, consider the contracts $\mathrm{I}[\sigma_1] = \{\mathtt{VoteA}, \mathtt{VoteB}\}[\mathtt{VoteA}]$ and $\mathrm{I}[\sigma_2] = \{\mathtt{VoteA}, \mathtt{VoteB}\}[\mathtt{VoteA} + \mathtt{VoteB}]$ and the client $\mathrm{K}[\rho] = \{\overline{\mathtt{VoteA}}, \overline{\mathtt{VoteB}}, \mathtt{e}\}[\overline{\mathtt{VoteA}}.\mathtt{e} + \overline{\mathtt{VoteB}}.\overline{\mathtt{VoteB}}.\mathtt{e}]$. Such client tries to vote for candidate `A` once or for `B` twice. It easy to verify that $\mathrm{K}[\rho]$ successfully interacts with $\mathrm{I}[\sigma_1]$ whilst the interaction may fail with $\mathrm{I}[\sigma_2]$, therefore $\mathrm{I}[\sigma_1] \not\preceq \mathrm{I}[\sigma_2]$. In contracts width subtyping is admitted provided the additional capabilities are not present in the interface of the smaller contract. For example $\{\mathtt{VoteA}\}[\mathtt{VoteA}] \preceq \{\mathtt{VoteA}, \mathtt{VoteB}\}[\mathtt{VoteA} + \mathtt{VoteB}]$ (the client $\mathrm{K}[\rho]$ is not a valid client for $\{\mathtt{VoteA}\}[\mathtt{VoteA}]$ because it has a larger interface). Similar arguments may be given for depth subtyping, the property guaranteeing the safety of the replacement of a service with another one providing a *longer* communication protocol. In the other direction, the differences follow by the fact that session types embody the property that sessions are supposed to be completed *symmetrically* by both parties, whilst contracts are biased towards clients, which are free to interrupt the interaction any time they please. Another difference is that contracts describe a more abstract synchronization pattern than session types do. For example, in the theory of contracts, a client such as

$$\{\overline{\mathtt{Login}}, \mathtt{Ok}, \overline{\mathtt{VoteA}}, \overline{\mathtt{VoteB}}, \mathtt{e}\}[\overline{\mathtt{Login}}.\mathtt{Ok}.(\overline{\mathtt{VoteA}}.\mathtt{e} + \overline{\mathtt{VoteB}}.\mathtt{e})]$$

may successfully interact with a service as $\textsc{i}[\tau]$, while, in session types, a branch cannot be matched by another branch. That is, session types describe a communication in which no handshaking between the interacting parties ever occurs. There is always exactly one party having control, and this party has to explicitly notify the other one about the (internal) choices it has made.

In this contribution we undertake a thorough comparison between session types and contracts for assessing a precise relationship between the two formalisms. We define two encodings, one from session types to contracts, and the other from contracts to session types. These encodings allows one to use the two formalisms interchangeably, without losing any relevant information. Therefore it is possible to argue about session types by means of the subcontract relation and, conversely, about contracts by using the deductive system of the subtyping relation. However, because of the differences between the two theories, the two encodings are not one the converse of the other. Let us discuss this issue with few examples. We encode the session type

$$T' \stackrel{\text{def}}{=} \&\langle \mathtt{Login} : \oplus\langle \mathtt{Ok} : \mathtt{end}\,;\ \mathtt{Cheat} : \mathtt{end}\rangle\rangle$$

into the contract

$$\textsc{i}'[\tau'] \stackrel{\text{def}}{=} [\mathtt{Login}, \overline{\mathtt{Ok}}, \overline{\mathtt{Cheat}}](\mathtt{Login}.(\overline{\mathtt{Ok}} \oplus \overline{\mathtt{Cheat}}) + \overline{\mathtt{Ok}}.\boldsymbol{\Omega} + \overline{\mathtt{Cheat}}.\boldsymbol{\Omega})$$

where the terms $\overline{\mathtt{Ok}}.\boldsymbol{\Omega}$ and $\overline{\mathtt{Cheat}}.\boldsymbol{\Omega}$ have been added in order to enforce width subtyping in the contract. If a client of $\textsc{i}'[\tau']$ attempts actions that are not explicitly allowed by T' then a catastrophic state – $\boldsymbol{\Omega}$ – is reached, meaning that, in practice, such actions are *not* guaranteed.

To illustrate the encoding of contracts into session types we discuss the encoding of $\{\mathtt{VoteA}, \mathtt{VoteB}\}[\mathtt{VoteA} \oplus \mathtt{VoteB}]$, which eventually generates the session type

$$\begin{array}{l}
\oplus\langle \{\mathtt{VoteA}\} : \&\langle \emptyset : \mathtt{end}; \{\mathtt{VoteA}\} : \mathtt{end}\rangle; \\
\quad \{\mathtt{VoteB}\} : \&\langle \emptyset : \mathtt{end}; \{\mathtt{VoteB}\} : \mathtt{end}\rangle; \\
\quad \{\mathtt{VoteA}, \mathtt{VoteB}\} : \&\langle \emptyset : \mathtt{end}; \{\mathtt{VoteA}\} : \mathtt{end}; \{\mathtt{VoteB}\} : \mathtt{end}\rangle\rangle
\end{array}$$

where we use sets of actions as labels. This type manifests a blow up of the input contract that is needed for compiling the complex synchronization patterns of contracts. Indeed, in the theory of contracts, $\{\mathtt{VoteA}, \mathtt{VoteB}\}[\mathtt{VoteA} \oplus \mathtt{VoteB}]$ is equivalent to $\{\mathtt{VoteA}, \mathtt{VoteB}\}[\mathtt{VoteA} \oplus \mathtt{VoteB} \oplus (\mathtt{VoteA} + \mathtt{VoteB})]$. That is, an internal choice between two alternatives means that one *or possibly both* are available. The encoding of contracts has to model explicitly which alternative is taken by sending a notification to the partner.

Related work. The research on contracts was inspired by "CCS without τ's" [7] and by Hennessy's model of acceptance trees [11, 12]. Contracts are an alternative representation of acceptance trees. The relation $\preceq$ was first introduced in [3], albeit it suffered from the lack of a clean semantic characterization and from the fact that it was not transitive. The version of $\preceq$ used in this paper is the same as the one introduced in [14]. In fact, $\preceq$ resembles the must preorder (and it reduces to the must preorder when the interfaces are large enough), but it arises from a notion of compliance that significantly differs from the notion of "passing a test" in the testing framework [6] and that more realistically describes well-behaved clients of Web services. The version of $\preceq$ we work with is actually a stricter version of a more powerful subcontract relation that has been investigated in [4].

Session types have been originally proposed in [13] and subsequently extended for dealing with functional languages [15], asynchrony, object-orientation [9]. In this paper we take [10] as the main reference for session types because it focuses on the subtyping relation. It is worth to notice that we restrict our analysis on the control aspects of session types, whilst other features such as first-class sessions and name passing, which are described in [10], have not been investigated in the framework of contracts yet.

Structure of the paper. We present the formal syntax and semantics of contracts in Section 2 and of session types in Section 3. Sections 4 and 5 present the encoding from session types to contracts and from contracts to session types, respectively. Section 6 concludes by summarizing the main similarities and differences between contracts and session types.

2 Contracts

The syntax of contracts uses an infinite set of *names* $\mathscr{N}$ ranged over by a, b, c, ..., and a disjoint set of *co-names* $\overline{\mathscr{N}}$ ranged over by $\overline{a}, \overline{b}, \overline{c}, \ldots$. Names and co-names are generically called *actions*. We let $\overline{\overline{a}} = a$ and use $\alpha, \beta, \ldots$ to range over actions; we let $\mathrm{I}, \mathrm{J}, \mathrm{K}, \ldots$ and $\mathrm{R}, \mathrm{S}, \ldots$ to range over (finite) sets of actions and we extend the operation $\bar{\cdot}$ to sets of actions so that $\overline{\mathrm{R}} = \{\overline{\alpha} \mid \alpha \in \mathrm{R}\}$. An infinite set of variables is also used, which is ranged over by $x, y, z, \ldots$.

Contracts are pairs $\mathrm{I}[\sigma]$ where I is a finite subset of $\mathscr{N} \cup \overline{\mathscr{N}}$ representing the *static interface* of the contract (all the actions occurring in σ must also occur in I), whereas σ, called *behavior*, is defined by the grammar:

$$\sigma \quad ::= \quad \mathbf{0} \quad | \quad \alpha.\sigma \quad | \quad \sigma \oplus \sigma \quad | \quad \sigma + \sigma \quad | \quad x \quad | \quad \texttt{rec}\ x.\sigma$$

Informally, $\mathbf{0}$ describes the inactive behavior; $\alpha.\sigma$ describes the behavior that performs an action α and then behaves like σ; $\sigma \oplus \tau$ describes the behavior that autonomously decides whether to behave as σ or as τ; $\sigma + \tau$ describes the behavior that lets the environment choose whether it should behave as σ or as τ; finally, $\texttt{rec}\ x.\sigma$ describes a recursive behavior that is equivalent to $\sigma\{^{\texttt{rec}\ x.\sigma}/_{x}\}$. In the following we write $\mathbf{\Omega}$ for the behavior $\texttt{rec}\ x.x$.

Behaviors retain a *transition relation* that is inductively defined by the rules

$$\alpha.\sigma \xrightarrow{\alpha} \sigma \qquad \sigma \oplus \tau \longrightarrow \sigma \qquad \frac{\sigma \xrightarrow{\alpha} \sigma'}{\sigma + \tau \xrightarrow{\alpha} \sigma'} \qquad \frac{\sigma \longrightarrow \sigma'}{\sigma + \tau \longrightarrow \sigma' + \tau}$$

$$\mathtt{rec}\; x.\sigma \longrightarrow \sigma\{^{\mathtt{rec}\; x.\sigma}/_{x}\}$$

We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$; $\sigma \overset{\alpha}{\Longrightarrow} \sigma'$ for $\sigma \Longrightarrow\xrightarrow{\alpha}\Longrightarrow \sigma'$; $\sigma \overset{\alpha}{\Longrightarrow}$ if there exists σ' such that $\sigma \overset{\alpha}{\Longrightarrow} \sigma'$. We write $\sigma\!\uparrow$ if σ has an infinite internal computation $\sigma = \sigma_0 \longrightarrow \sigma_1 \longrightarrow \sigma_2 \longrightarrow \cdots$ and $\sigma\!\downarrow$ if not $\sigma\!\uparrow$. We write $\sigma\!\downarrow\!\alpha_1 \cdots \alpha_n$ if $\sigma\!\downarrow$ and, if $\sigma \overset{\alpha_1}{\Longrightarrow} \sigma'$ implies $\sigma'\!\downarrow\!\alpha_2 \cdots \alpha_n$; we write $\sigma\!\uparrow\!\varphi$ otherwise. For example $\mathbf{\Omega}\!\uparrow$, $\mathtt{rec}\; x.a + x\!\uparrow$, and $\mathtt{rec}\; x.(a.x + b.x)\!\downarrow\!\varphi$ for every $\varphi \in \{a, b\}^*$. We let $\mathtt{init}(\sigma)$ be $\{\alpha \mid \sigma \overset{\alpha}{\Longrightarrow}\}$.

A basic use of contracts is to verify whether a client protocol is *compliant with* a service protocol. This compliance is possible if, independently of the internal choices of both client and service, the client successfully completes every interaction with the service. We now formalize the notions of "interaction" and of "successful completion":

- The interaction of a client and a service is defined by the relation $\longrightarrow$ over pairs of behaviors as follows:

  $$\frac{\rho \longrightarrow \rho'}{\rho \parallel \sigma \longrightarrow \rho' \parallel \sigma} \qquad \frac{\sigma \longrightarrow \sigma'}{\rho \parallel \sigma \longrightarrow \rho \parallel \sigma'} \qquad \frac{\rho \xrightarrow{\alpha} \rho' \quad \sigma \xrightarrow{\overline{\alpha}} \sigma'}{\rho \parallel \sigma \longrightarrow \rho' \parallel \sigma'}$$

  where we assume that ρ is a client contract and σ is a service contract. As usual we write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$.
- The successful completion of the client is modeled using a special name $\mathtt{e}$. The client has successfully completed the interaction with the service if no further synchronization with the service is possible and the client can emit an $\mathtt{e}$ action. We assume that behaviors never manifest co-names $\overline{\mathtt{e}}$.

Definition 1 (Compliance). *Let* $\mathtt{e} \notin \textsc{s}$. *The (client) contract* $\textsc{k}[\rho]$ *is* compliant with *the (service) contract* $\textsc{i}[\sigma]$*, written* $\textsc{k}[\rho] \dashv \textsc{i}[\sigma]$*, if* $\overline{\textsc{k} \setminus \{\mathtt{e}\}} \subseteq \textsc{i}$ *and* $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma'$ *implies*

1. *if* $\rho' \parallel \sigma' \not\longrightarrow$*, then* $\{\mathtt{e}\} \subseteq \mathtt{init}(\rho')$*;*
2. *if* $\sigma'\!\uparrow$*, then* $\{\mathtt{e}\} = \mathtt{init}(\rho')$*.*

According to the notion of behavioral compliance, if a client $\textsc{k}[\rho]$ is compliant with a service $\textsc{i}[\sigma]$ then it should never attempt to perform actions that are not allowed by the interface of the service it is interacting with. If the client-service conversation terminates, then the client is in a successful state (it will emit $\mathtt{e}$). For example, $a.\mathtt{e} + b.\mathtt{e} \dashv \overline{a} \oplus \overline{b}$ and $a.\mathtt{e} \oplus b.\mathtt{e} \dashv \overline{a} + \overline{b}$ but $a.\mathtt{e} \oplus b.\mathtt{e} \not\dashv \overline{a} \oplus \overline{b}$ because of the computation $a.\mathtt{e} \oplus b.\mathtt{e} \parallel \overline{a} \oplus \overline{b} \Longrightarrow a.\mathtt{e} \parallel \overline{b} \not\longrightarrow$ where the client waits for an interaction on a in vain. Similarly, the client must reach a successful state if the conversation does not terminate but the divergence is due to the service. In this case, however, the client cannot rely on any signal from the service, not even an end-of-connection one, so it is required to do nothing but terminate.

Following De Nicola and Hennessy's approach to process semantics [6], this test induces a preorder on services on the basis of the set of clients that comply with a given service.

Definition 2 (Subcontract). *A contract* $\mathrm{I}[\sigma]$ *is a* subcontract *of* $\mathrm{J}[\tau]$*, written* $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$*, if and only if, for every* $\mathrm{K}[\rho]$*, we have* $\mathrm{K}[\rho] \dashv \mathrm{I}[\sigma]$ *implies* $\mathrm{K}[\rho] \dashv \mathrm{J}[\tau]$*. We let* $\mathrm{I}[\sigma] \simeq \mathrm{J}[\tau]$ *if both* $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ *and* $\mathrm{J}[\tau] \preceq \mathrm{I}[\sigma]$*.*

That is, if a client is compliant with a service $\mathrm{I}[\sigma]$ and $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$, then the same client is also compliant with $\mathrm{J}[\tau]$. Hence, the service $\mathrm{J}[\tau]$ can be safely used where $\mathrm{I}[\sigma]$ is expected. As usual it is easier to figure out inequalities: $\{a,b\}[a] \not\preceq \{a,b\}[a.b]$ because $\{\overline{a},\overline{b},\mathsf{e}\}[\overline{a}.(\mathsf{e}+\overline{b})] \dashv \{a,b\}[a]$ but $\{\overline{a},\overline{b},\mathsf{e}\}[\overline{a}.(\mathsf{e}+\overline{b})] \not\dashv \{a,b\}[a.b]$; $\{a,b\}[a] \not\preceq \{a,b\}[a+b]$ because $\{\overline{b},\mathsf{e}\}[\mathsf{e}+\overline{b}] \dashv \{a,b\}[a]$ but $\{\overline{b},\mathsf{e}\}[\mathsf{e}+\overline{b}] \not\dashv \{a,b\}[a+b]$.

Since the set of clients compliant with a given service is usually infinite, Definition 2 gives little insight on the properties of $\preceq$. This calls for a direct, coinductive characterization of $\preceq$, which also happens to be easier to work with in the proofs of the results that follow.

Definition 3 (Coinductive subcontract). *Let* $\sigma \Downarrow \mathrm{R}$ *if and only if* $\sigma \Longrightarrow \sigma'$ *and* $\mathrm{R} = \mathtt{init}(\sigma')$*. The relation* $\mathscr{R}$ *is a* coinductive subcontract *if* $\mathrm{I}[\sigma] \mathrel{\mathscr{R}} \mathrm{J}[\tau]$ *implies* $\mathrm{I} \subseteq \mathrm{J}$ *and whenever* $\sigma\downarrow$ *then*

1. $\tau\downarrow$*, and*
2. $\tau \Downarrow \mathrm{R}$ *implies* $\sigma \Downarrow \mathrm{R}'$ *and* $\mathrm{R}' \subseteq \mathrm{R}$*, and*
3. $\alpha \in \mathrm{I}$ *and* $\tau \stackrel{\alpha}{\Longrightarrow} \tau'$ *implies that there exist* $\sigma_1, \dots, \sigma_n$ *such that* $\sigma \stackrel{\alpha}{\Longrightarrow} \sigma_i$ *for every* $1 \leq i \leq n$ *and* $\mathrm{I}[\bigoplus_{1 \leq i \leq n} \sigma_i] \mathrel{\mathscr{R}} \mathrm{J}[\tau']$*.*

By this definition, a contract $\mathrm{I}[\sigma]$ such that $\sigma\uparrow$ is the smallest one with interface I. When $\sigma\downarrow$, condition 1 constrains the larger contract $\mathrm{J}[\tau]$ to converge as well, since clients might rely on the convergence of σ to complete successfully. Condition 2 states that $\mathrm{J}[\tau]$ must exhibit a more deterministic behavior: the smaller the number of ready sets is, the more deterministic the contract is. Furthermore, $\mathrm{J}[\tau]$ should expose *at least* the same capabilities as the smaller one ($\mathrm{R}' \subseteq \mathrm{R}$). Condition 3 is perhaps the most subtle one, as it deals with all the possible derivatives of the smaller contract. The point is that $\{a,b,c\}[a.b + a.c] \simeq \{a,b,c\}[a.(b \oplus c)]$ since, after interacting on a, a client of the service on the left side of $\simeq$ is not aware of which state the service is in (it can be either b or c). Hence, we have to consider all of the possible derivatives after a, thus reducing to verifying $\{a,b,c\}[(b+c) \oplus b \oplus c] \mathrel{\mathscr{R}} \{a,b,c\}[b \oplus c]$ which trivially holds.

The set-theoretic and the coinductive versions of the subcontract relation do coincide, as the following proposition states (a proof can be found in the full version of [14]).

Proposition 1. $\preceq$ *is the largest coinductive subcontract.*

The theory of $\preceq$ has been thoroughly studied in [14]; in particular it has been put in correspondence with a well-known equivalence – the *must testing* [12]. A useful property relating $\preceq$ and $\longrightarrow$ is in order.

Proposition 2. *If* $\sigma \Longrightarrow \sigma'$*, then* $\mathrm{I}[\sigma] \preceq \mathrm{I}[\sigma']$*.*

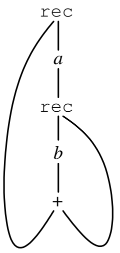
For every $\mathrm{I}[\sigma]$ there exists σ' in normal form such that $\mathrm{I}[\sigma] \simeq \mathrm{I}[\sigma']$. This result is slightly more general than those in [5, 6, 12], where normal forms are used to demonstrate the completeness of an axiomatization and are defined for recursion-free terms only. The normal form uses particular families of sets of actions – the *acceptance set*.

Definition 4 (acceptance set [12]). *The* acceptance set *of a behavior* σ*, denoted by* $\mathscr{A}(\sigma)$*, is defined as* $\mathscr{A}(\sigma) \stackrel{\text{def}}{=} \{\mathtt{R} \mid \mathtt{R} \subseteq \mathtt{init}(\sigma)$ *and* $\exists \mathtt{S} \subseteq \mathtt{R} : \sigma \Downarrow \mathtt{S}\}$*. We let* $\mathscr{A}$*,* $\mathscr{A}'$*, ... range over acceptance sets.*

Definition 5 (normal form). *A behavior* σ *is in* normal form *if either* $\sigma = \mathbf{\Omega}$ *or* $\sigma = x$ *or* $\sigma = \bigoplus_{\mathtt{R} \in \mathscr{A}(\sigma)} \sum_{\alpha \in \mathtt{R}} \alpha.\sigma_\alpha$ *or* $\sigma = \mathtt{rec}\ x. \bigoplus_{\mathtt{R} \in \mathscr{A}(\sigma)} \sum_{\alpha \in \mathtt{R}} \alpha.\sigma_\alpha$ *and each* σ_α *is in normal form.*

Let $\sigma(\alpha) \stackrel{\text{def}}{=} \bigoplus_{\sigma \Longrightarrow \xrightarrow{\alpha} \sigma'} \sigma'$ and let $\mathtt{nf}(\sigma)$ be a family of behavior names defined as follows

$$\mathtt{nf}(\sigma) \stackrel{\text{def}}{=} \begin{cases} \mathbf{\Omega} & \text{if } \sigma\uparrow \\ \bigoplus_{\mathtt{R} \in \mathscr{A}(\sigma)} \sum_{\alpha \in \mathtt{R}} \alpha.\mathtt{nf}(\sigma(\alpha)) & \text{otherwise} \end{cases}$$



It is not obvious that $\sigma(\alpha)$ is well defined and that behavior names $\mathtt{nf}(\sigma)$ may be folded into a finite behavior (by using recursion and variables). In order to prove these facts, we represent behaviors as syntax trees where variables are pointers to the corresponding binder [1]. For example, the figure on the right shows the syntax tree of $\sigma = \mathtt{rec}\ x.a.\mathtt{rec}\ y.b.(x + y)$.

As usual, every node in syntax trees corresponds to a closed term (i.e. a behavior) that is unique up to the name of bound variables. For example, in the above tree, the node b corresponds to the behavior $b.(\sigma + (\mathtt{rec}\ y.b.(\sigma + y)))$. The formal definition of behavior associated with a node is omitted because standard.

Lemma 1. *Let* $\lfloor \sigma \rfloor \stackrel{\text{def}}{=} \{\tau \mid \exists \alpha : \alpha.\tau$ *occurs in the syntax tree of* $\sigma\}$*. If* $\sigma \stackrel{\varphi}{\Longrightarrow} \sigma'$*, then* $\lfloor \sigma' \rfloor \subseteq \lfloor \sigma \rfloor$*.*

Proof. We prove that the property holds by induction on the derivation of $\sigma \longrightarrow \sigma'$ or $\sigma \xrightarrow{\alpha} \sigma'$ (symmetric cases are omitted). The lemma follows by a straightforward induction on the length of the derivation $\sigma \stackrel{\varphi}{\Longrightarrow} \sigma'$.

($\sigma = \alpha.\sigma' \xrightarrow{\alpha} \sigma'$) We distinguish two subcases: if σ does occur in σ', then $\lfloor \sigma' \rfloor = \lfloor \sigma \rfloor$; if σ does *not* occur in σ', then $\lfloor \sigma' \rfloor = \lfloor \sigma \rfloor \setminus \{\sigma'\} \subset \lfloor \sigma \rfloor$.

($\sigma = \sigma' \oplus \tau \longrightarrow \sigma'$) We conclude immediately $\lfloor \sigma' \rfloor \subseteq \lfloor \sigma' \oplus \tau \rfloor = \lfloor \sigma \rfloor$.

($\sigma = \tau + \tau', \tau \xrightarrow{\alpha} \sigma'$) By induction hypothesis we have $\lfloor \sigma' \rfloor \subseteq \lfloor \tau \rfloor$ hence $\lfloor \sigma' \rfloor \subseteq \lfloor \tau \rfloor \subseteq \lfloor \tau + \tau' \rfloor = \lfloor \sigma \rfloor$.

($\sigma = \tau + \tau'', \tau \longrightarrow \tau', \sigma' = \tau' + \tau''$) By induction hypothesis we have $\lfloor \tau' \rfloor \subseteq \lfloor \tau \rfloor$, hence $\lfloor \sigma' \rfloor = \lfloor \tau' + \tau'' \rfloor \subseteq \lfloor \tau + \tau'' \rfloor = \lfloor \sigma \rfloor$.

($\sigma = \mathtt{rec}\ x.\tau, \sigma \longrightarrow \tau\{^{\sigma}/_{x}\} = \sigma'$) We conclude immediately $\lfloor \sigma' \rfloor = \lfloor \sigma \rfloor$. ☐

It is worth to notice that Lemma 1 does not hold if behaviors are extended with a parallel operator "|". For example if $\sigma = \mathtt{rec}\ x.a \mid (b.x + 0)$, then $\sigma(b) = a \mid \sigma$, which is not a subtree in the syntax tree of σ.

Lemma 2. *Let* $D(\sigma) \stackrel{\text{def}}{=} \{\sigma(\varphi) \mid \sigma \stackrel{\varphi}{\Longrightarrow}\}$*. Then* $D(\sigma)$ *is finite.*

Proof. Let $D_0, D_1, \ldots$ be the family of sets defined as follows:

$$D_0 \stackrel{\text{def}}{=} \{\sigma\} \qquad D_{i+1} \stackrel{\text{def}}{=} D_i \cup \{\sigma' \mid \exists \sigma \in D_i : \exists \alpha : \sigma \overset{\alpha}{\Longrightarrow\!\!\longrightarrow} \sigma'\}$$

We only need to show that there exists n such that $D_n = D_{n+1}$, because each $\sigma(\varphi)$ is obtained by joining some continuations σ', for some subtrees $\alpha.\sigma'$ in σ. Let n be the cardinality of $\lfloor \sigma \rfloor$ (n is the number of *distinct* subtrees in the syntax tree of σ and it exists because σ is finite). By contradiction, assume that there exists $\sigma_{n+1} \in D_{n+1} \backslash D_n$. Then there exist $\sigma_1, \ldots, \sigma_n$ and $\alpha_1, \ldots, \alpha_n, \alpha_{n+1}$ such that

$$\sigma \overset{\alpha_1}{\Longrightarrow\!\!\longrightarrow} \sigma_1 \overset{\alpha_2}{\Longrightarrow\!\!\longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow\!\!\longrightarrow} \sigma_n \overset{\alpha_{n+1}}{\Longrightarrow\!\!\longrightarrow} \sigma_{n+1}$$

and $\sigma_i \in D_i \setminus D_{i-1}$ for every $1 \leq i \leq n$. We have $\sigma_{n+1} \in \lfloor \sigma_n \rfloor$. By Lemma 1 we deduce $\sigma_{n+1} \in \lfloor \sigma \rfloor$. Since $\lfloor \sigma \rfloor$ has only n distinct subtrees, σ_{n+1} is reachable from σ with at most n reductions, thus $\sigma_{n+1} \in D_n$, which contradicts $\sigma_{n+1} \in D_{n+1} \setminus D_n$. ☐

Theorem 1. *For every* $\mathrm{I}[\sigma]$ *there exists* σ' *in normal form such that* $\mathrm{I}[\sigma] \simeq \mathrm{I}[\sigma']$. *The behavior* σ' *is the folding of* $\mathtt{nf}(\sigma)$.

Proof. By Lemma 2 there are finitely many $\sigma(\varphi)$. These $\sigma(\varphi)$ are in one-to-one correspondence with the behavior names that may be recursively invoked by $\mathtt{nf}(\sigma)$. Therefore the set of such behavior names is finite and $\mathtt{nf}(\sigma)$ may be folded into a behavior with recursion and variables.

To prove the subcontract equivalence, let $\mathscr{R}$ be the least relation containing $\preceq$ and such that

1. if $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ and τ' is the normal form of τ, then $\mathrm{I}[\sigma] \mathrel{\mathscr{R}} \mathrm{J}[\tau']$;
2. if $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ and σ' is the normal form of σ, then $\mathrm{I}[\sigma'] \mathrel{\mathscr{R}} \mathrm{J}[\tau]$.

We prove that $\mathscr{R}$ is a subcontract relation; the theorem follows directly.

Let $\mathrm{I}[\sigma] \mathrel{\mathscr{R}} \mathrm{J}[\tau']$, where τ' is the normal form of τ. Then $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$. As regards condition 1 in the definition of coinductive subcontract, notice that from $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ we have $\sigma\!\downarrow$ implies $\tau\!\downarrow$, hence $\tau'\!\downarrow$ from the definition of normal form. As regards condition 2, let $\tau' \Downarrow \mathrm{R}$, then $\mathrm{R} \in \mathscr{A}(\tau)$, hence $\tau \Downarrow \mathrm{R}'$ and $\mathrm{R}' \subseteq \mathrm{R}$ by definition of $\mathscr{A}(\tau)$. From $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ we obtain $\sigma \Downarrow \mathrm{S}$ and $\mathrm{S} \subseteq \mathrm{R}'$, hence we conclude $\mathrm{S} \subseteq \mathrm{R}$. As regards condition 3, let $\tau' \overset{\alpha}{\Longrightarrow} \tau''$. Then $\mathrm{J}[\tau'(\alpha)] \preceq \mathrm{J}[\tau'']$ and, by definition, $\tau'(\alpha)$ is in normal form. Then $\tau \overset{\alpha}{\Longrightarrow}$ and, by $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$, $\sigma \overset{\alpha}{\Longrightarrow}$. Therefore $\mathrm{I}[\sigma(\alpha)] \preceq \mathrm{J}[\tau(\alpha)]$, hence by definition of $\mathscr{R}$, $\mathrm{I}[\sigma(\alpha)] \mathrel{\mathscr{R}} \mathrm{J}[\tau''']$, where τ''' is the normal form of $\tau(\alpha)$. By definition of normal form $\tau''' = \tau'(\alpha)$ and we conclude.

Let $\mathrm{I}[\sigma'] \mathrel{\mathscr{R}} \mathrm{J}[\tau]$, where σ' is the normal form of σ. Then $\mathrm{I}[\sigma] \mathrel{\mathscr{R}} \mathrm{J}[\tau]$. As regards condition 1 in the definition of coinductive subcontract, $\sigma'\!\downarrow$ implies $\sigma\!\downarrow$, hence $\tau\!\downarrow$. As regards condition 2, let $\tau \Downarrow \mathrm{R}$. From $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ we have $\sigma \Downarrow \mathrm{S}$ and $\mathrm{S} \subseteq \mathrm{R}$ and we conclude by observing that $\sigma' \Downarrow \mathrm{S}$. As regards condition 3, let $\tau \overset{\alpha}{\Longrightarrow} \tau'$. Then there exist $\sigma_1, \ldots, \sigma_n$ such that $\mathrm{I}[\bigoplus_{1 \leq i \leq n} \sigma_i] \preceq \mathrm{J}[\tau']$. Since $\mathrm{I}[\sigma(\alpha)] \preceq \mathrm{I}[\bigoplus_{1 \leq i \leq n} \sigma_i]$ and the normal form of $\sigma(\alpha)$ is equal to $\sigma'(\alpha)$ we conclude $\mathrm{I}[\sigma'(\alpha)] \mathrel{\mathscr{R}} \mathrm{J}[\tau']$ by definition of $\mathscr{R}$. ☐

By Theorem 1, from now on we let $\mathtt{nf}(\sigma)$ be the normal form of σ.

3 Session types

Similarly to a contract, a session type is meant to describe the type of a communication as a sequence of actions and branching points. The syntax of session types uses an infinite set of labels $\mathscr{L}$ ranged over by $\ell, \ell', \ldots$, and an infinite set of variables ranged over by $x, y, z, \ldots$. Session types are defined by the grammar:

$$S \quad ::= \quad \texttt{end} \quad | \quad \oplus\langle \ell_i : S_i{}^{i\in I}\rangle \quad | \quad \&\langle \ell_i : S_i{}^{i\in I}\rangle \quad | \quad x \quad | \quad \mu x.S$$

The session type end describes a completed communication; the session type $\oplus\langle l_i : S_i{}^{i\in I}\rangle$ describes a *choice* where a process autonomously decides to proceed according to one of the continuations S_i's. Before doing so, the process notifies the partner of the communication by sending a label ℓ_i; the session type $\&\langle \ell_i : S_i{}^{i\in I}\rangle$ describes a *branch* where a process is ready to proceed according to any of the continuations S_i's. Before doing so, the process waits for a label ℓ_i from the process it is interacting with. The label uniquely identifies the continuation. The session type $\mu x.S$ describes a recursive type, much like a recursive contract.

Following [10], session types are taken *contractive*, namely every occurrence of a variable is guarded by at least a choice or a branch. Unlike [10], however, we omit session types describing the communications of actual data (not merely labels) during a session. Typically, such communications are represented by session types of the form $!t.S$ (for "send a message of type t on the channel and then continue as S") or $?t.S$ (for "receive a message of type t from the channel and then continue as S"). However, as far as the comparison of session types and contracts is concerned, the presence of such actions is not particularly relevant, since we are going to focus on the control part of an interaction. The addition of $!t$ and $?t$ actions to the above grammar does not pose any particular problem for all of the results that follow, but at the same time it does not contribute any significant insight in the comparison.

The semantics of session types is defined in terms of a subtyping relation $\leq$, which is presented by means of a deductive system or as a coinductive definition [10]. In this contribution we stick to this latter presentation that turns out to be easier to compare with the subcontract relation defined in the previous section. More precisely, $S \leq T$, whenever every process with type S can be used where a process with type T is expected (in this respect, $\leq$ differs from $\preceq$ as it is oriented as a real subtyping relation). Because of the structure of session types, $\leq$ is quite straightforward to define: basically every choice in S must be defined on a subset of the labels occurring in the corresponding choice in T, whereas any branch in T must be defined on a superset of the labels occurring in the corresponding branch in T. Let

$$\texttt{unfold}(S) \stackrel{\text{def}}{=} \begin{cases} \texttt{unfold}(T\{^S/_x\}) & \text{if } S = \mu x.T \\ S & \text{otherwise} \end{cases}$$

That is, $\texttt{unfold}(\cdot)$ removes topmost recursions until it reveals a branch or a choice. It is worth to notice that $\texttt{unfold}(\cdot)$ is always defined because session types are contractive.

Definition 6 (Subtyping). *The relation $\mathscr{R}$ is a* coinductive subtyping *if $S \mathrel{\mathscr{R}} T$ implies*

1. *if* $\texttt{unfold}(S) = \texttt{end}$*, then* $\texttt{unfold}(T) = \texttt{end}$*,*

2. *if* $\mathtt{unfold}(S) = \oplus\langle \ell_i : S_i{}^{i\in I}\rangle$*, then* $\mathtt{unfold}(T) = \oplus\langle \ell_j : T_j{}^{j\in J}\rangle$ *and* $I \subseteq J$ *and for every* $i \in I$ *we have* $S_i \mathscr{R} T_i$*,*
3. *if* $\mathtt{unfold}(S) = \&\langle \ell_i : S_i{}^{i\in I}\rangle$*, then* $\mathtt{unfold}(T) = \&\langle \ell_j : T_j{}^{j\in J}\rangle$ *and* $J \subseteq I$ *and for every* $j \in J$ *we have* $S_j \mathscr{R} T_j$*.*

Let $\leq$ *be the largest coinductive subtyping.*

For example, we have $\oplus\langle \ell : S\rangle \leq \oplus\langle \ell : S; \ell' : T\rangle$ since the smaller session type represents a process that behaves more deterministically than the one represented by the larger session type. Dually we have $\&\langle \ell : S; \ell' : T\rangle \leq \&\langle \ell : S\rangle$ since the process represented by the smaller session type offers a wider range of continuations (it is ready to accept a superset of the labels accepted by the process represented by the larger session type). Unlike contracts, however, we have $\&\langle \ell : S\rangle \not\leq \mathtt{end}$ and $\oplus\langle \ell : S\rangle \not\leq \mathtt{end}$. In the first case the process *waits for* a label that never arrives, in the second case it *sends* a label that the matching party is not ready to receive.

Session types are naturally equipped with a $\mathtt{dual}(\cdot)$ function computing the dual type.

$$\begin{aligned}
\mathtt{dual}(\mathtt{end}) &= \mathtt{end}\\
\mathtt{dual}(\oplus\langle \ell_i : S_i{}^{i\in I}\rangle) &= \&\langle \ell_i : \mathtt{dual}(S_i)^{\ i\in I}\rangle\\
\mathtt{dual}(\&\langle \ell_i : S_i{}^{i\in I}\rangle) &= \oplus\langle \ell_i : \mathtt{dual}(S_i)^{\ i\in I}\rangle\\
\mathtt{dual}(x) &= x\\
\mathtt{dual}(\mu x.S) &= \mu x.\mathtt{dual}(S)
\end{aligned}$$

As for $\mathtt{unfold}(\cdot)$, $\mathtt{dual}(\cdot)$ is always defined because session types are contractive. If S is the session type representing the behavior of a process, then $\mathtt{dual}(S)$ represents the behavior of a "canonical" process that interacts successfully with the first one. The theory of session session types does not formalize the notion of successful interaction, but we will be able to reason about it by means of one of our encodings in Section 5.

The duality between choices and branches in session types is formalized by the following proposition. Let $\mathtt{in}(S)$ (respectively, $\mathtt{out}(S)$) be the set of labels occurring in branches (respectively, choices) of S. Then subtyping induces a relation on labels $\mathtt{in}(\cdot)$ and $\mathtt{out}(\cdot)$.

Proposition 3. *If* $S \leq T$ *then (1)* $\mathtt{out}(S) \subseteq \mathtt{out}(T)$*, (2)* $\mathtt{in}(T) \subseteq \mathtt{in}(S)$*, and (3)* $\mathtt{dual}(T) \leq \mathtt{dual}(S)$*.*

Notice that from the previous proposition and from the fact that $\mathtt{dual}(\cdot)$ is the inverse of itself ($\mathtt{dual}(\mathtt{dual}(S)) = S$), we have $S \leq T$ if and only if $\mathtt{dual}(T) \leq \mathtt{dual}(S)$.

4 Encoding session types into contracts

In order to encode session types into contracts we preliminarily need to set a correspondence between labels and names. For simplicity we let $\mathscr{L} = \mathscr{N}$. Therefore, in the following, we will not distinguish between labels and names and we will address them with ℓ, a, $\ldots$.

$$
\begin{aligned}
\llbracket \mathsf{end} \rrbracket_{\textsc{i}} &= \sigma_e \\
\llbracket \oplus\langle \ell_i : S_i{}^{i\in 1..n}\rangle \rrbracket_{\textsc{i}} &= \textstyle\bigoplus_{i\in 1..n} \overline{\ell_i}.\llbracket S_i \rrbracket_{\textsc{i}} \\
\llbracket \&\langle \ell_i : S_i{}^{i\in 1..n}\rangle \rrbracket_{\textsc{i}} &= \textstyle\sum_{i\in 1..n} \ell_i.\llbracket S_i \rrbracket_{\textsc{i}} + \sum_{\alpha\in\textsc{i}\setminus\{\ell_1,\dots,\ell_n\}} \alpha.\mathbf{\Omega} \\
\llbracket x \rrbracket_{\textsc{i}} &= x \\
\llbracket \mu x.S \rrbracket_{\textsc{i}} &= \mathtt{rec}\ x.\llbracket S \rrbracket_{\textsc{i}}
\end{aligned}
$$

Table 1. Encoding of session types into contracts.

The encoding $\llbracket S \rrbracket_{\textsc{i}}$ of a session type S is defined with respect to an interface $\textsc{i}$ (Table 1). The terminal behavior σ_e is $\mathtt{e}$ if the session type being encoded regards a client, or **0** if it regards a service. The only case that deserves discussion is branch. The contract $\llbracket \&\langle \ell_i : S_i{}^{i\in 1..n}\rangle \rrbracket_{\textsc{i}}$ has as many initial actions as are allowed by the interface, irrespective of the labels of the branches. This enforces width subtyping in the resulting contracts, which is otherwise false, in general. In facts, $\{a, b\}[a] \not\preceq \{a, b\}[a+b]$ because the client $\{\overline{a}, \overline{b}\}[\overline{a}.\mathtt{e} + \overline{b}]$ is compliant with the first service, but not with the second one. By translating $S = \&\langle a : S'\rangle$ as simply $\{a, b\}[a.\llbracket S' \rrbracket]$ (the action b is in the interface because it occurs in S'), then a client such as $\{\overline{a}, \overline{b}\}[\overline{a}.\mathtt{e} + \overline{b}.\rho]$ could fail when one replaces $\llbracket S \rrbracket_{\{a,b\}}$ with $\llbracket T \rrbracket_{\{a,b\}}$ and $T \leq S$, for instance when $T = \&\langle a : S' \,;\, b : T'\rangle$. To avoid these cases, one has to exclude $\{\overline{a}, \overline{b}\}[\overline{a}.\mathtt{e} + \overline{b}.\rho]$ from those clients that are compliant with $\llbracket S \rrbracket$ because it attempts to do an action ($\overline{b}$) that is not explicitly allowed by S. This exclusion follows by translating a branch into an external choice "+" with a summand for every action in the interface: the actions that do not appear as labels of the branch are given a continuation $\mathbf{\Omega}$. Therefore, the contract $\llbracket S \rrbracket_{\{a,b\}}$ does not admit, in general, a client $\{\overline{a}, \overline{b}\}[\overline{a}.\mathtt{e} + \overline{b}.\rho]$ but only a client $\{\overline{a}, \overline{b}\}[\overline{a}.\mathtt{e} + \overline{b}.\mathtt{e}]$ (this client has to terminate after an interaction on b, without requiring any further capability to the service). The contract $\mathbf{\Omega}$ is the smallest one with a given interface: a client compliant with $\mathbf{\Omega}$ will also comply with any other service, in particular with those resulting from the encoding of smaller session types.

Example 1. According to the encoding, we have

- $\llbracket \&\langle a : \mathsf{end}\rangle \rrbracket_{\{a\}} = a$;
- $\llbracket \&\langle a : \mathsf{end}\rangle \rrbracket_{\{a,b\}} = a + b.\mathbf{\Omega}$;
- $\llbracket \&\langle a : \mathsf{end}; b : \mathsf{end}\rangle \rrbracket_{\{a,b\}} = a + b$;
- $\llbracket \oplus\langle a : \mathsf{end}\rangle \rrbracket_{\{a\}} = \llbracket \oplus\langle a : \mathsf{end}\rangle \rrbracket_{\{a,b\}} = \overline{a}$;
- $\llbracket \oplus\langle a : \mathsf{end}; b : \mathsf{end}\rangle \rrbracket_{\{a,b\}} = \overline{a} \oplus \overline{b}$.

We notice that $\&\langle a : \mathsf{end}; b : \mathsf{end}\rangle \leq \&\langle a : \mathsf{end}\rangle$ and $\{a\}[a] \preceq \{a, b\}[a + b.\mathbf{\Omega}] \preceq \{a, b\}[a+b]$. Similarly, $\oplus\langle a : \mathsf{end}\rangle \leq \oplus\langle a : \mathsf{end}; b : \mathsf{end}\rangle$ and $\{a, b\}[\overline{a}\oplus\overline{b}] \preceq \{a, b\}[\overline{a}]$. ■

More generally, the encoding of Table 1 makes the subtyping relation and the subcontract relation agree.

Theorem 2. *Let* $\textsc{j} = \mathtt{in}(T) \cup \overline{\mathtt{out}(T)}$ *and* $\textsc{i} = \mathtt{in}(S) \cup \overline{\mathtt{out}(T)}$. *Then* $S \leq T$ *if and only if* $\textsc{j}[\llbracket T \rrbracket_{\textsc{j}}] \preceq \textsc{i}[\llbracket S \rrbracket_{\textsc{i}}]$.

Proof. ("only if" part) Let $\mathscr{R}$ be the least relation such that

1. if $\textsc{j} \subseteq \textsc{i}$ then $\textsc{j}[\mathbf{\Omega}] \mathrel{\mathscr{R}} \textsc{i}[\sigma]$;
2. if $S \leq T$ and $\textsc{j} = \mathtt{in}(T) \cup \overline{\mathtt{out}(T)}$ and $\textsc{i} = \mathtt{in}(S) \cup \overline{\mathtt{out}(T)}$, then $\textsc{j}[\llbracket T \rrbracket_{\textsc{j}}] \mathrel{\mathscr{R}} \textsc{i}[\llbracket S \rrbracket_{\textsc{i}}]$.

We prove that $\mathscr{R}$ is a coinductive subcontract. Let $\textsc{j}[\llbracket T \rrbracket_{\textsc{j}}] \mathrel{\mathscr{R}} \textsc{i}[\llbracket S \rrbracket_{\textsc{i}}]$. By Proposition 3, $\textsc{j} \subseteq \textsc{i}$. By definition, $\llbracket \cdot \rrbracket_{\cdot}$ always yields behaviors σ such that $\sigma{\downarrow}$. Therefore condition 1 in Definition 3 is trivially satisfied. As regards conditions 2 and 3, we reason by cases on S. By definition of $\leq$, $S \leq T$ if and only if $\mathtt{unfold}(S) \leq \mathtt{unfold}(T)$. Therefore we may restrict to cases where neither S nor T are recursive types.

($S = \mathtt{end}$) Then $T = \mathtt{end}$. Condition 2 follows immediately; condition 3 is vacuous.

($S = \oplus\langle \ell_h : S_h \;{}^{h \in H}\rangle$) Then $T = \oplus\langle \ell_k : T_k \;{}^{k \in K}\rangle$ and $H \subseteq K$ and $S_h \leq T_h$, for every $h \in H$. As regards condition 2, if $\llbracket S \rrbracket_{\textsc{i}} \Downarrow \textsc{r}$ then $\overline{\ell_h} \in \textsc{r}$ for some $h \in H$. Since $H \subseteq K$, $\llbracket T \rrbracket_{\textsc{j}} \Downarrow \{\overline{\ell_h}\}$ and $\{\overline{\ell_h}\} \subseteq \textsc{r}$. As regards conditions 3, by Proposition 2, it suffices to consider the transitions $\llbracket S \rrbracket_{\textsc{i}} \overset{\overline{\ell_h}}{\Longrightarrow} \llbracket S_h \rrbracket_{\textsc{i}}$ for every $h \in H$. Since $H \subseteq K$, $\llbracket T \rrbracket_{\textsc{j}} \overset{\overline{\ell_h}}{\Longrightarrow} \llbracket T_h \rrbracket_{\textsc{j}}$ and $\llbracket T_h \rrbracket_{\textsc{j}} \mathrel{\mathscr{R}} \llbracket S_h \rrbracket_{\textsc{i}}$ follows by $S_h \leq T_h$;

($S = \&\langle \ell_k : S_k \;{}^{k \in K}\rangle$) Then $T = \&\langle \ell_h : T_h \;{}^{h \in H}\rangle$ with $H \subseteq K$. Condition 2 follows because $\llbracket T \rrbracket_{\textsc{j}} \Downarrow \{\ell_h \mid h \in H\}$ and $\llbracket S \rrbracket_{\textsc{i}} \Downarrow \{\ell_k \mid k \in K\}$ and $\{\ell_h \mid h \in H\} \subseteq \{\ell_k \mid k \in K\}$. As regards condition 3, by Proposition 2, it suffices to consider $\llbracket S \rrbracket_{\textsc{i}} \overset{\ell_k}{\Longrightarrow} \llbracket S_k \rrbracket_{\textsc{i}}$ with $k \in \textsc{j}$. There are two subcases: either (i) $\ell_k \notin \{\ell_h \mid h \in H\}$ or (ii) $\ell_k \in \{\ell_h \mid h \in H\}$. In subcase (i), $\llbracket T \rrbracket_{\textsc{j}} \overset{\ell_k}{\Longrightarrow} \mathbf{\Omega}$ and $\textsc{j}[\mathbf{\Omega}] \mathrel{\mathscr{R}} \textsc{i}[\llbracket S_k \rrbracket_{\textsc{i}}])$ by definition of $\mathscr{R}$. In subcase (ii), $\llbracket T \rrbracket_{\textsc{j}} \overset{\ell_k}{\Longrightarrow} \llbracket T_k \rrbracket_{\textsc{j}}$. From $S_k \leq T_k$ we derive $\textsc{j}[\llbracket T_k \rrbracket_{\textsc{j}}] \mathrel{\mathscr{R}} \textsc{i}[\llbracket S_k \rrbracket_{\textsc{i}}]$ by definition of $\mathscr{R}$.

("if" part) We prove that if $S \not\leq T$, then $\textsc{j}[\llbracket T \rrbracket_{\textsc{j}}] \not\preceq \textsc{i}[\llbracket S \rrbracket_{\textsc{i}}]$. We reason by cases on the shape of S and T. It is sufficient to consider those cases in which none of the types begins with a recursion and $S \leq T$ holds directly, without looking at the session types in the choices and branches possibly found in S and T.

($S = \mathtt{end}$) It must be $T \neq \mathtt{end}$. Then $\llbracket S \rrbracket_{\textsc{i}} \Downarrow \{\mathtt{e}\}$ whereas $\llbracket T \rrbracket_{\textsc{j}} \Downarrow \textsc{r}$ implies $\textsc{r} \neq \emptyset$ and $\mathtt{e} \notin \textsc{r}$;

($S = \oplus\langle \ell_i : S_i \;{}^{i \in I}\rangle$) If $T = \mathtt{end}$, then we can reason as for the case $S = \mathtt{end}$ and conclude that $\textsc{j}[\llbracket T \rrbracket_{\textsc{j}}] \not\preceq \textsc{i}[\llbracket S \rrbracket_{\textsc{i}}]$. If $T = \&\langle \ell_j : T_j \;{}^{j \in J}\rangle$, then $\llbracket T \rrbracket_{\textsc{j}}$ has only one ready set $\textsc{j} = \mathtt{in}(T) \cup \overline{\mathtt{out}(T)}$, which contains at least two actions and hence cannot be smaller than $\{\ell_i\}$, for every $i \in I$. If $T = \oplus\langle \ell_j : T_j \;{}^{j \in J}\rangle$ and $I \not\subseteq J$, then there exists $i \in I$ such that $i \notin J$. So $\llbracket S \rrbracket_{\textsc{i}} \Downarrow \{\overline{\ell_i}\}$ whereas $\llbracket T \rrbracket_{\textsc{j}} \Downarrow \textsc{r}$ implies $\textsc{r} \neq \emptyset$ and $\overline{\ell_i} \notin \textsc{r}$;

($S = \&\langle \ell_i : S_i \;{}^{i \in I}\rangle$) If $T = \mathtt{end}$ we conclude immediately since no ℓ_i is equal to $\mathtt{e}$. If $T = \oplus\langle \ell_j : T_j \;{}^{j \in J}\rangle$, then from $\textsc{i} = \mathtt{in}(S) \cup \overline{\mathtt{out}(T)}$ we have that $j \in J$ implies $\overline{\ell_j} \in \textsc{i}$. Now we have $\llbracket S \rrbracket_{\textsc{i}} \overset{\overline{\ell_j}}{\Longrightarrow} \mathbf{\Omega}$, whereas $\llbracket T \rrbracket_{\textsc{j}} \overset{\overline{\ell_i}}{\Longrightarrow} \sigma$ implies $\sigma{\downarrow}$. If $T = \&\langle \ell_j : T_j \;{}^{j \in J}\rangle$ and $J \not\subseteq I$, then there exists $j \in J$ such that $j \notin I$. If $\textsc{j} \not\subseteq \textsc{i}$ there is nothing to prove. If $\textsc{j} \subseteq \textsc{i}$, then $\llbracket S \rrbracket_{\textsc{i}} \overset{\overline{\ell_j}}{\Longrightarrow} \mathbf{\Omega}$, whereas $\llbracket T \rrbracket_{\textsc{j}} \overset{\overline{\ell_j}}{\Longrightarrow} \sigma$ implies $\sigma{\downarrow}$. □

The following proposition shows that internal moves in the encoding of a session type S correspond to reducing its choices and determining a session type $S' \leq S$.

Proposition 4. *If* $\llbracket S \rrbracket_{\mathrm{I}} \Longrightarrow \sigma$*, then* $\sigma = \llbracket S' \rrbracket_{\mathrm{I}}$ *and* $S' \leq S$*.*

Proof. It is sufficient to consider the case $\llbracket S \rrbracket_{\mathrm{I}} \longrightarrow \sigma$, then the lemma follows directly. We reason by cases on the structure of S, there are only two possibilities:

($S = \oplus\langle \ell_i : S_i \,^{i \in K} \rangle$) Then $\llbracket S \rrbracket_{\mathrm{I}} = \bigoplus_{i \in I} \overline{\ell_i}.\llbracket S_i \rrbracket_{\mathrm{I}} \longrightarrow \bigoplus_{i \in J} \overline{\ell_j}.\llbracket S_j \rrbracket_{\mathrm{I}} = \llbracket \oplus\langle \ell_j : S_j \,^{j \in J} \rangle \rrbracket_{\mathrm{I}}$ with $J \subseteq I$. We conclude by observing that $\oplus\langle \ell_j : S_j \,^{j \in J} \rangle \leq S$.

($S = \mu x.S'$) Then $\llbracket S \rrbracket_{\mathrm{I}} = \texttt{rec}\ x.\llbracket S' \rrbracket_{\mathrm{I}} \longrightarrow \llbracket S' \rrbracket_{\mathrm{I}}\{^{\texttt{rec}\ x.\llbracket S' \rrbracket_{\mathrm{I}}}/_{x}\} = \llbracket S'\{^{S}/_{x}\} \rrbracket_{\mathrm{I}}$ and this is the only possible reduction. We conclude because $S'\{^{S}/_{x}\} \leq S$. ☐

The encoding in Table 1 allows us to relate session types and their duals. The relation is exactly the compliance of Definition 1.

Theorem 3. *Let* $\mathrm{I} = \texttt{in}(S) \cup \overline{\texttt{out}(S)}$ *and* $\mathrm{J} = \texttt{in}(T) \cup \overline{\texttt{out}(T)}$ *and* $S \leq \texttt{dual}(T)$*. Then* $\llbracket S \rrbracket_{\mathrm{I}} \dashv \llbracket T \rrbracket_{\mathrm{J}}$*.*

Proof. Let $\mathscr{R}$ be the least relation such that if $S \leq \texttt{dual}(T)$ and $\texttt{in}(S) \cup \overline{\texttt{out}(S)} \subseteq \mathrm{I}$ and $\texttt{in}(T) \cup \overline{\texttt{out}(T)} \subseteq \mathrm{J}$, then $\llbracket S \rrbracket_{\mathrm{I}} \mathrel{\mathscr{R}} \llbracket T \rrbracket_{\mathrm{J}}$. We prove that $\mathscr{R}$ is a compliance relation. Let $\llbracket S \rrbracket_{\mathrm{I}} \mathrel{\mathscr{R}} \llbracket T \rrbracket_{\mathrm{J}}$. By Proposition 4, if $\llbracket S \rrbracket_{\mathrm{I}} \longrightarrow \rho$, then $\rho = \llbracket S' \rrbracket_{\mathrm{I}}$ and $S' \leq S \leq \texttt{dual}(T)$. Symmetrically, if $\llbracket T \rrbracket_{\mathrm{J}} \longrightarrow \sigma$, then $\sigma = \llbracket T' \rrbracket_{\mathrm{J}}$ and $T' \leq T$, hence, by Proposition 3(3), $S \leq \texttt{dual}(T) \leq \texttt{dual}(T')$. It follows that we may restrict our analysis to cases where neither the encoding of S nor the encoding of T can perform internal moves:

($S = \texttt{end}$) Then $T = \llbracket \texttt{end} \rrbracket_{\mathrm{J}}$. Therefore $\llbracket S \rrbracket_{\mathrm{I}} \parallel \llbracket T \rrbracket_{\mathrm{J}} \not\longrightarrow$ and $\llbracket S \rrbracket_{\mathrm{I}} \xrightarrow{\texttt{e}}$;

($S = \oplus\langle \ell : S' \rangle$) Then $T = \&\langle \ell_i : T_i \,^{i \in K} \rangle_{\mathrm{J}}$ and $\ell = \ell_i$ for some $i \in K$ and $S' \leq \texttt{dual}(T_i)$. Now $\llbracket S \rrbracket_{\mathrm{I}} \xrightarrow{\overline{\ell}}$ and $\llbracket T \rrbracket_{\mathrm{J}} \xrightarrow{\ell}$, hence $\llbracket S \rrbracket_{\mathrm{I}} \parallel \llbracket T \rrbracket_{\mathrm{J}} \longrightarrow \llbracket S' \rrbracket_{\mathrm{I}} \parallel \llbracket T_i \rrbracket_{\mathrm{J}}$ and we conclude $\llbracket S' \rrbracket_{\mathrm{I}} \mathrel{\mathscr{R}} \llbracket T_i \rrbracket_{\mathrm{J}}$ by definition of $\mathscr{R}$.

($S = \&\langle \ell_i : S_i \,^{i \in I} \rangle$) Dual of the previous case. ☐

5 Encoding contracts into session types

The encoding of contracts to session types is partially defined. In particular, we will restrict the encoding to behaviors that are convergent. The reason for this restriction derives from the fact that session types describe communications whose end is agreed by both parties having type `end`. The behavior $\mathbf{\Omega}$, on the other hand, represents a service that may not pay any attention to the communication it is involved in, and it has no corresponding session type.

A behavior σ is *strongly convergent* if, for every sequence φ, $\sigma\downarrow\varphi$. The following encodings are defined on contracts with *strongly convergent* behaviors. It is also convenient to restrict the encoding of client contracts to those whose behavior never leads to $\mathbf{0}$ without emitting `e`. For example, the behavior $a.\texttt{e} + b.\mathbf{0}$ describes a client that succeeds if the service proposes $\overline{a}$, but that fails if the service proposes $\overline{b}$. Such a behavior has no counterpart in session types, where failures are not explicitly represented.

In general, if a client is unable to handle a particular action, like b in the example, it should simply omit that action from its behavior. We say that a (client) contract $\mathtt{I}[\rho]$ is *canonical* if, whenever $\rho \overset{\varphi}{\Longrightarrow} \rho'$ is maximal, then $\varphi = \varphi'\mathtt{e}$ and $\mathtt{e} \notin \mathtt{names}(\varphi')$. For example $\{a\}[a.\mathtt{e}]$, $\{a\}[\mathtt{rec}\ x.a.x]$, and $\emptyset[\mathbf{\Omega}]$ are canonical; $\{a, b\}[a.\mathtt{e} + b.\mathbf{0}]$ and $\{a\}[\mathtt{rec}\ x.a + x]$ are not canonical.

The encoding of contracts into session types is reported in Table 2. We write $\mathtt{S} \bowtie \mathscr{A}$ ($\mathscr{A}$ is an acceptance set) if for every $\mathtt{R} \in \mathscr{A}$, either $\mathtt{e} \in \mathtt{R}$ or $\mathtt{S} \cap \mathtt{R} \neq \emptyset$. We use an injective map $\widehat{\cdot}$ from sets of actions to labels that we leave unspecified.

Encoding of client contracts/behaviors:

$$
\begin{aligned}
\mathscr{C}[\![\mathtt{K}[\rho]]\!] &= \oplus\langle \widehat{\mathtt{K} \setminus \{\mathtt{e}\}} : \mathscr{C}[\![\mathtt{nf}(\rho)]\!]_{\mathtt{K}\setminus\{\mathtt{e}\}} \rangle \\
\mathscr{C}[\![\textstyle\bigoplus_{\mathtt{R}\in\mathscr{A}} \sum_{\alpha\in\mathtt{R}} \alpha.\rho_\alpha]\!]_{\mathtt{K}} &= \&\langle \widehat{\overline{\mathtt{S}}} : \oplus\langle \underbrace{\widehat{\emptyset} : \mathtt{end};}_{\text{if } \mathtt{e} \in \mathtt{R}} \widehat{\{\overline{\alpha}\}} : \mathscr{C}[\![\rho_\alpha]\!]_{\mathtt{K}}{}^{\mathtt{R}\in\mathscr{A},\alpha\in\mathtt{S}\cap\mathtt{R}} \rangle^{\mathtt{S}\subseteq\overline{\mathtt{K}},\mathtt{S}\bowtie\mathscr{A}} \rangle \\
\mathscr{C}[\![\mathtt{rec}\ x.\rho]\!]_{\mathtt{K}} &= \mu x.\mathscr{C}[\![\rho]\!]_{\mathtt{K}} \\
\mathscr{C}[\![x]\!]_{\mathtt{K}} &= x
\end{aligned}
$$

Encoding of service contracts/behaviors:

$$
\begin{aligned}
\mathscr{S}[\![\mathtt{I}[\sigma]]\!] &= \&\langle \widehat{\mathtt{K}} : \mathscr{S}[\![\mathtt{nf}(\sigma)]\!]_{\mathtt{K}}{}^{\mathtt{K}\subseteq\mathtt{I}} \rangle \\
\mathscr{S}[\![\textstyle\bigoplus_{\mathtt{R}\in\mathscr{A}} \sum_{\alpha\in\mathtt{R}} \alpha.\sigma_\alpha]\!]_{\mathtt{K}} &= \oplus\langle \widehat{\mathtt{R} \cap \mathtt{K}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{\alpha\}} : \mathscr{S}[\![\sigma_\alpha]\!]_{\mathtt{K}}{}^{\alpha\in\mathtt{R}\cap\mathtt{K}} \rangle^{\mathtt{R}\in\mathscr{A}} \rangle \\
\mathscr{S}[\![\mathtt{rec}\ x.\rho]\!]_{\mathtt{K}} &= \mu x.\mathscr{S}[\![\rho]\!]_{\mathtt{K}} \\
\mathscr{S}[\![x]\!]_{\mathtt{K}} &= x
\end{aligned}
$$

Table 2. Encoding of contracts into session types.

The encoding distinguishes between client and service contracts. One reason is that, unlike sessions, which are supposed to be completed *symmetrically* by both parties, the theory of contracts is biased towards clients, which are free to interrupt the interaction any time they please. The other reason is that contracts describe a more abstract synchronization pattern than session types do, and the encoding of contracts into session types has to render this synchronization pattern, which amounts to a little handshaking protocol.

The session types corresponding to the client contract $\mathtt{K}[\rho]$ and the service contract $\mathtt{I}[\sigma]$ are denoted by $\mathscr{C}[\![\mathtt{K}[\rho]]\!]$ and $\mathscr{S}[\![\mathtt{I}[\sigma]]\!]$, respectively. The labels in the types represent finite sets of actions in the source contracts. Let us discuss the encoding of a service contract. The first step in the generation of the session type is the offering of an interface to the client. Any client that asks no more capabilities than those offered by the service can connect. Hence, the service offers as many interfaces as the number of the subsets of its own interface. For every offered interface, the continuation session type is a specialized encoding of the service's contract, restricted to the offered interface: $\mathscr{S}[\![\sigma]\!]_{\mathtt{K}}$. For the proper encoding of the behavior, we resort to the normal form (Definition 5). A behavior of the form $\bigoplus_{\mathtt{R}\in\mathscr{A}} \sum_{\alpha\in\mathtt{R}} \alpha.\sigma_\alpha$ is one in which the service can be in as many states as the cardinality of $\mathscr{A}$. In each state $\textsc{r} \in \mathscr{A}$, the service is ready to perform any of the actions in $\textsc{r}$. So, the service begins by communicating to the client the state it is in (restricted to the interface of the connected client). Then, the service accepts a singleton action, among those that are available, indicating the choice of the client, or the special action $\widehat{\emptyset}$ denoting the fact that the client has decided to terminate at this stage. We notice that the term

$$\oplus\langle \widehat{\textsc{r} \cap \textsc{k}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{\alpha\}} : \mathscr{S}[\![\sigma_\alpha]\!]_{\textsc{k}}{}^{\alpha \in \textsc{r} \cap \textsc{k}} \rangle^{\textsc{r} \in \mathscr{A}} \rangle$$

is well formed: every label of a branch or a choice has exactly one continuation because the behavior $\bigoplus_{\textsc{r} \in \mathscr{A}} \sum_{\alpha \in \textsc{r}} \alpha.\sigma_\alpha$ is in normal form.

In some sense the encoding of a service contract is "kind" as it tries to accommodate the largest number of clients (not only those connecting with exactly the same interface as the service, but also with smaller ones). Conversely, the encoding of a client contract is selfish in that it only encodes the client behavior, whose only purpose is to successfully achieve its task. The first action of the client is the selection of the service interface that matches with its own (containing the co-actions of the client's interface). Then, at each interaction, the client must be ready to accept a set $\overline{\textsc{s}}$ of actions from the service, representing the state the service is in. However, not all such states are suitable for the client. In particular, let $\mathscr{A}$ be the acceptance set of the client; the client will only accept those states $\overline{\textsc{s}}$ such that, for every $\textsc{r} \in \mathscr{A}$, either $\mathtt{e} \in \textsc{r}$ (the client is ready to terminate) or $\textsc{s} \cap \textsc{r} \neq \emptyset$ (the client and the service can synchronize). Said otherwise, the fewer sets $\overline{\textsc{s}}$ the client can accept from the service, the more demanding it is. Once a set has been accepted, the client may choose the special label $\widehat{\emptyset}$, signaling its intention to terminate the interaction, or it may choose a label $\widehat{\{\alpha\}}$, signaling the intention of synchronizing on α. Because of the way the sets $\overline{\textsc{s}}$ are accepted by the client, it is always possible to pursue at least one of such possibilities.

There are discretions in the two encodings $\mathscr{C}[\![\cdot]\!]$ and $\mathscr{S}[\![\cdot]\!]$: the alternation between $\oplus$ and $\&$ may be reversed without changing the following correctness results.

Example 2. Consider the service contracts

$$\textsc{i}[\sigma] \stackrel{\text{def}}{=} \{a, b\}[a \oplus b] \qquad \text{and} \qquad \textsc{i}[\tau] \stackrel{\text{def}}{=} \{a, b\}[a + b]$$

and notice that $\textsc{i}[\sigma] \preceq \textsc{i}[\tau]$. Their respective encodings are

$$\begin{aligned}
\mathscr{S}[\![\textsc{i}[\sigma]]\!] = \&\langle \widehat{\emptyset} : {} & \oplus\langle \widehat{\emptyset} : \&\langle \widehat{\emptyset} : \mathtt{end}\rangle\rangle; \\
& \widehat{\{a\}} : \oplus\langle \widehat{\emptyset} : \&\langle \widehat{\emptyset} : \mathtt{end}\rangle; \widehat{\{a\}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{a\}} : \mathtt{end}\rangle\rangle; \\
& \widehat{\{b\}} : \oplus\langle \widehat{\emptyset} : \&\langle \widehat{\emptyset} : \mathtt{end}\rangle; \widehat{\{b\}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{b\}} : \mathtt{end}\rangle\rangle; \\
& \widehat{\{a, b\}} : \oplus\langle \widehat{\{a\}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{a\}} : \mathtt{end}\rangle; \\
& \qquad\qquad \widehat{\{b\}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{b\}} : \mathtt{end}\rangle; \\
& \qquad\qquad \widehat{\{a, b\}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{a\}} : \mathtt{end}; \widehat{\{b\}} : \mathtt{end}\rangle\rangle\rangle \\
\mathscr{S}[\![\textsc{i}[\tau]]\!] = \&\langle \widehat{\emptyset} : {} & \oplus\langle \widehat{\emptyset} : \&\langle \widehat{\emptyset} : \mathtt{end}\rangle\rangle; \\
& \widehat{\{a\}} : \oplus\langle \widehat{\{a\}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{a\}} : \mathtt{end}\rangle\rangle; \\
& \widehat{\{b\}} : \oplus\langle \widehat{\{b\}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{b\}} : \mathtt{end}\rangle\rangle; \\
& \widehat{\{a, b\}} : \oplus\langle \widehat{\{a, b\}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{a\}} : \mathtt{end}; \widehat{\{b\}} : \mathtt{end}\rangle\rangle\rangle
\end{aligned}$$

and now we have $\mathscr{S}\llbracket \mathrm{I}[\tau] \rrbracket \leq \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$. Notice that, in the encoding of $\mathrm{I}[\sigma]$, we have $\mathscr{A}(\sigma) = \{\{a\}, \{b\}, \{a, b\}\}$. ■

More generally, the encodings of related contracts are related session types, except that the direction of the preorder is reversed.

Theorem 4. *Let σ and τ strongly convergent. Then $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ if and only if $\mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket \leq \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$.*

Proof. ("only if" part) Let $\mathscr{R}$ be the least relation such that

- if $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ and σ and τ are strongly convergent, then $\mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket \ \mathscr{R}\ \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$; additionally, if $\mathrm{K} \subseteq \mathrm{I}$ then $\mathscr{S}\llbracket \mathtt{nf}(\tau) \rrbracket_{\mathrm{K}} \ \mathscr{R}\ \mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\mathrm{K}}$.

We prove that $\mathscr{R}$ is a coinductive subtyping. Let $S \ \mathscr{R}\ T$. We have two possibilities, according to the definition of $\mathscr{R}$.

1. ($S = \mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket$, $T = \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$, and $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$) By definition of $\mathscr{S}\llbracket \cdot \rrbracket$ we have $S = \&\langle \widehat{\mathrm{K}} : \mathscr{S}\llbracket \mathtt{nf}(\tau) \rrbracket_{\mathrm{K}} \ {}^{\mathrm{K} \subseteq \mathrm{J}} \rangle$ and $T = \&\langle \widehat{\mathrm{K}} : \mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\mathrm{K}} \ {}^{\mathrm{K} \subseteq \mathrm{I}} \rangle$. From $\mathrm{I} \subseteq \mathrm{J}$ we have $\{\widehat{\mathrm{K}} \mid \mathrm{K} \subseteq \mathrm{I}\} \subseteq \{\widehat{\mathrm{K}} \mid \mathrm{K} \subseteq \mathrm{J}\}$, hence each label in the topmost branch of T also occurs as a label in the topmost branch of S. Now take $\mathrm{K} \subseteq \mathrm{I}$. We have to show that $\mathscr{S}\llbracket \mathtt{nf}(\tau) \rrbracket_{\mathrm{K}} \ \mathscr{R}\ \mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\mathrm{K}}$), but this follows immediately from the definition of $\mathscr{R}$ and from $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$.
2. ($S = \mathscr{S}\llbracket \mathtt{nf}(\tau) \rrbracket_{\mathrm{K}}$, $T = \mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\mathrm{K}}$, $\mathrm{K} \subseteq \mathrm{I}$, and $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$) We have several subcases depending on the shape of the normal form of σ and τ. Assume that $\mathtt{nf}(\sigma) = \bigoplus_{\mathrm{R} \in \mathscr{A}(\sigma)} \sum_{\alpha \in \mathrm{R}} \alpha.\sigma_\alpha$ and $\mathtt{nf}(\tau) = \bigoplus_{\mathrm{S} \in \mathscr{A}(\tau)} \sum_{\beta \in \mathrm{S}} \beta.\tau_\beta$. Then $S = \oplus\langle \widehat{\mathrm{S} \cap \mathrm{K}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{\beta\}} : \mathscr{S}\llbracket \tau_\beta \rrbracket_{\mathrm{K}} \ {}^{\beta \in \mathrm{S} \cap \mathrm{K}} \rangle \ {}^{\mathrm{S} \in \mathscr{A}(\tau)} \rangle$ and $T = \oplus\langle \widehat{\mathrm{R} \cap \mathrm{K}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{\alpha\}} : \mathscr{S}\llbracket \sigma_\alpha \rrbracket_{\mathrm{K}} \ {}^{\alpha \in \mathrm{R} \cap \mathrm{K}} \rangle \ {}^{\mathrm{R} \in \mathscr{A}(\sigma)} \rangle$. From $\mathrm{I}[\sigma] \preceq \mathrm{J}[\tau]$ we have that $\{\mathrm{S} \cap \mathrm{K} \mid \mathrm{S} \in \mathscr{A}(\tau)\} \subseteq \{\mathrm{R} \cap \mathrm{K} \mid \mathrm{R} \in \mathscr{A}(\sigma)\}$, hence each label in the topmost choice of S also occurs as a label in the topmost choice of T. Take $\mathrm{S} \in \mathscr{A}(\tau)$. The label $\widehat{\mathrm{S} \cap \mathrm{K}}$ occurs in both S and T, hence the corresponding branches have exactly the same set of labels $\{\widehat{\emptyset}\} \cup \{\widehat{\{\alpha\}} \mid \alpha \in \mathrm{S} \cap \mathrm{K}\}$. Let $S' = \mathscr{S}\llbracket \tau_\alpha \rrbracket_{\mathrm{K}}$ and $T' = \mathscr{S}\llbracket \sigma_\alpha \rrbracket_{\mathrm{K}}$. From $\mathrm{I}[\mathtt{nf}(\sigma)] \preceq \mathrm{J}[\mathtt{nf}(\tau)]$ and $\alpha \in \mathrm{I}$ we have that $\mathtt{nf}(\tau) \xRightarrow{\alpha}$ implies $\mathtt{nf}(\sigma) \xRightarrow{\alpha}$ and $\mathrm{I}[\sigma_\alpha] \preceq \mathrm{J}[\tau_\alpha]$, so we conclude $S' \ \mathscr{R}\ T'$ by definition of $\mathscr{R}$. Assume that $\mathtt{nf}(\sigma) = \mathtt{rec}\, x.\sigma'$, where σ' is in normal form. Then $\mathrm{I}[\sigma] \simeq \mathrm{I}[\sigma'\{{}^{\mathtt{nf}(\sigma)}/_{x}\}]$ and $\sigma'\{{}^{\mathtt{nf}(\sigma)}/_{x}\}$ is itself in normal form. Since σ is strongly convergent, we eventually reach some σ'' in normal form such that σ'' does not begin with a recursion and $\mathrm{I}[\sigma] \simeq \mathrm{I}[\sigma'']$. Similarly for τ. Hence, we easily reduce to the previous subcase.

("if" part) We prove that $\mathrm{I}[\sigma] \not\preceq \mathrm{J}[\tau]$ implies $\mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket \not\leq \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$. It suffices to consider all the possibilities by which $\mathrm{I}[\sigma] \not\preceq \mathrm{J}[\tau]$ directly:

1. if $\mathrm{I} \not\subseteq \mathrm{J}$, then there exists $\alpha \in \mathrm{I}$ such that $\alpha \notin \mathrm{J}$. Then $\widehat{\{\alpha\}}$ is a label occurring in the topmost branch of $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$ but not occurring in the topmost branch of $\mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket$, hence $\mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket \not\leq \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$;
2. let $\mathrm{R}_1, \dots, \mathrm{R}_n$ be the ready sets of σ and assume that there exists R such that $\tau \Downarrow \mathrm{R}$ and for every $1 \leq i \leq n$ we have $\mathrm{R}_i \not\subseteq \mathrm{R}$, that is for every $1 \leq i \leq n$ there exists

$\alpha_i \in \mathrm{R}_i \cap \mathrm{I}$ and $\alpha_i \notin \mathrm{R}$. From $\mathrm{I} \subseteq \mathrm{J}$ we know that both $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$ and $\mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket$ have the label $\widehat{\mathrm{I}}$ in their corresponding topmost branches with continuations $\mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\mathrm{I}}$ and $\mathscr{S}\llbracket \mathtt{nf}(\tau) \rrbracket_{\mathrm{I}}$ respectively. Then, by the fact that $\widehat{\cdot}$ is injective, it follows that $\widehat{\mathrm{R} \cap \mathrm{I}} \notin \{\widehat{\mathrm{R}_i \cap \mathrm{I}} \mid \mathrm{R}_i \in \mathscr{A}(\sigma)\}$, hence $\widehat{\mathrm{R} \cap \mathrm{I}}$ is a label occurring in the topmost choice of $\mathscr{S}\llbracket \mathtt{nf}(\tau) \rrbracket_{\mathrm{I}}$ but not occurring in the topmost choice of $\mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\mathrm{I}}$, so we conclude $\mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket \not\leq \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$;

3. assume that there exists $\alpha \in \mathrm{I}$ such that $\tau \overset{\alpha}{\Longrightarrow}$ and $\sigma \overset{\alpha}{\not\Longrightarrow}$. Then $\tau \Downarrow \mathrm{S}$ where $\alpha \in \mathrm{S}$ whereas $\sigma \Downarrow \mathrm{R}$ implies $\alpha \notin \mathrm{R}$. Hence we can reason as for the previous case and conclude $\mathscr{S}\llbracket \mathrm{J}[\tau] \rrbracket \not\leq \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$. $\square$

The strict correspondence between $\leq$ and $\preceq$ allows one to use them interchangeably, as discussed in the example below.

Example 3. Consider the client contracts

$$\mathrm{K}[\rho] \overset{\text{def}}{=} \{\bar{a}, \bar{b}, \mathsf{e}\}[\bar{a}.\mathsf{e} + \bar{b}.\mathsf{e}] \qquad \text{and} \qquad \mathrm{K}[\rho'] \overset{\text{def}}{=} \{\bar{a}, \bar{b}, \mathsf{e}\}[\bar{a}.\mathsf{e} \oplus \bar{b}.\mathsf{e}]$$

and notice that $\mathrm{K}[\rho] \dashv \mathrm{I}[\sigma]$ whereas $\mathrm{K}[\rho'] \not\dashv \mathrm{I}[\sigma]$, where $\mathrm{I}[\sigma]$ is the service contract defined in Example 2. The respective encodings of these two client contracts are

$$\begin{aligned}
\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket = \oplus\langle \widehat{\{a,b\}} : \&\langle & \widehat{\{a\}} : \oplus\langle \widehat{\{a\}} : \mathtt{end}\rangle \\
& \widehat{\{b\}} : \oplus\langle \widehat{\{b\}} : \mathtt{end}\rangle \\
& \widehat{\{a,b\}} : \oplus\langle \widehat{\{a\}} : \mathtt{end}; \widehat{\{b\}} : \mathtt{end}\rangle\rangle\rangle \\
\mathscr{C}\llbracket \mathrm{K}[\rho'] \rrbracket = \oplus\langle \widehat{\{a,b\}} : \&\langle & \widehat{\{a,b\}} : \oplus\langle \widehat{\{a\}} : \mathtt{end}; \widehat{\{b\}} : \mathtt{end}\rangle\rangle\rangle
\end{aligned}$$

and now we notice that $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket \leq \mathtt{dual}(\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket)$, namely it is safe to use $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$ to interact successfully with $\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket$. On the other hand $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket \not\leq \mathtt{dual}(\mathscr{C}\llbracket \mathrm{K}[\rho'] \rrbracket)$ because $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$ may be in a state where only a or only b are available, whilst the client $\mathrm{K}[\rho']$ autonomously decides which of the two actions to execute. Notice however that $\mathscr{S}\llbracket \mathrm{I}[\tau] \rrbracket \leq \mathtt{dual}(\mathscr{C}\llbracket \mathrm{K}[\rho'] \rrbracket)$. ■

More generally, if $\mathrm{K}[\rho] \dashv \mathrm{I}[\sigma]$ holds then it is safe to use the service $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket$ to interact with the client $\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket$ and in fact $\mathtt{dual}(\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket)$ is the *principal service type* that interacts successfully with $\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket$.

Theorem 5. $\mathrm{K}[\rho] \dashv \mathrm{I}[\sigma]$ *if and only if* $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket \leq \mathtt{dual}(\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket)$.

Proof. ("only if" part) Let $\mathscr{R}$ be the least relation such that

- if $\mathrm{K}[\rho] \dashv \mathrm{I}[\sigma]$ and ρ and σ are strongly convergent, then $\mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket \mathrel{\mathscr{R}} \mathtt{dual}(\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket)$ and $\mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\overline{\mathrm{K}\setminus\{\mathtt{e}\}}} \mathrel{\mathscr{R}} \mathtt{dual}(\mathscr{C}\llbracket \mathtt{nf}(\rho) \rrbracket_{\mathrm{K}\setminus\{\mathtt{e}\}})$.

It is sufficient to show that $\mathscr{R}$ is a coinductive subtyping. Let $S \mathrel{\mathscr{R}} T$. We have two possibilities.

1. ($S = \mathscr{S}\llbracket \mathrm{I}[\sigma] \rrbracket, T = \mathtt{dual}(\mathscr{C}\llbracket \mathrm{K}[\rho] \rrbracket)$, and $\mathrm{K}[\rho] \dashv \mathrm{I}[\sigma]$) Then $S = \&\langle \widehat{\mathrm{H}} : \mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\mathrm{H}}{}^{\mathrm{H} \subseteq \mathrm{I}}\rangle$ and $T = \&\langle \widehat{\overline{\mathrm{K} \setminus \{\mathtt{e}\}}} : \mathtt{dual}(\mathscr{C}\llbracket \mathtt{nf}(\rho) \rrbracket_{\mathrm{K}\setminus\{\mathtt{e}\}})\rangle$. From $\overline{\mathrm{K} \setminus \{\mathtt{e}\}} \subseteq \mathrm{I}$ we have that the label in the topmost branch of T also occurs as a label in the topmost label of S. We must show that $\mathscr{S}\llbracket \mathtt{nf}(\sigma) \rrbracket_{\mathrm{H}} \mathrel{\mathscr{R}} \mathtt{dual}(\mathscr{C}\llbracket \mathtt{nf}(\rho) \rrbracket_{\mathrm{K}\setminus\{\mathtt{e}\}})$, but this is obvious by definition of $\mathscr{R}$.

2. $(S = \mathscr{S}\llbracket \mathtt{nf}(\sigma)\rrbracket_{\overline{\mathrm{K}}}, T = \mathtt{dual}(\mathscr{C}\llbracket \mathtt{nf}(\rho)\rrbracket_{\mathrm{K}})$, and $(\mathrm{K} \cup \{\mathtt{e}\})[\rho] \dashv \mathrm{I}[\sigma])$ Let $\mathtt{nf}(\sigma) = \bigoplus_{\mathrm{R}\in\mathscr{A}(\sigma)} \sum_{\alpha\in\mathrm{R}} \alpha.\sigma_\alpha$ and $\mathtt{nf}(\rho) = \bigoplus_{\mathrm{R}\in\mathscr{A}(\rho)} \sum_{\alpha\in\mathrm{R}} \alpha.\rho_\alpha$. Then

$$\begin{aligned} S &= \oplus\langle \widehat{\mathrm{R}\cap\overline{\mathrm{K}}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{\alpha\}} : \mathscr{S}\llbracket \mathtt{nf}(\sigma_\alpha)\rrbracket_{\overline{\mathrm{K}}} {}^{\alpha\in\mathrm{R}\cap\overline{\mathrm{K}}}\rangle {}^{\mathrm{R}\in\mathscr{A}(\sigma)}\rangle \\ T &= \oplus\langle \widehat{\overline{\mathrm{S}}} : \&\langle \widehat{\emptyset} : \mathtt{end}; \widehat{\{\overline{\alpha}\}} : \mathtt{dual}(\mathscr{C}\llbracket \mathtt{nf}(\rho_\alpha)\rrbracket_{\mathrm{K}}) {}^{\mathrm{R}\in\mathscr{A}(\rho),\alpha\in\mathrm{S}\cap\mathrm{R}}\rangle {}^{\mathrm{S}\subseteq\mathrm{K},\mathrm{S}\bowtie\mathscr{A}(\rho)}\rangle \end{aligned}$$

Let $\mathrm{R} \in \mathscr{A}(\sigma)$ be a ready set of the service. Then $\overline{\mathrm{R}\cap\overline{\mathrm{K}}} = \overline{\mathrm{R}}\cap\mathrm{K} \subseteq \mathrm{K}$. Furthermore, by definition of $\bowtie$, we have that $\overline{\mathrm{R}\cap\overline{\mathrm{K}}} \bowtie \mathscr{A}(\rho)$ if and only if for every $\mathrm{R}' \in \mathscr{A}(\rho)$ we have either $\mathtt{e} \in \mathrm{R}'$ or $\overline{\mathrm{R}\cap\overline{\mathrm{K}}}\cap\mathrm{R}' \neq \emptyset$. But $\mathrm{R}' \subseteq \mathrm{K}$, hence $\overline{\mathrm{R}\cap\overline{\mathrm{K}}}\cap\mathrm{R}' = \overline{\mathrm{R}}\cap\mathrm{R}'$. So $\overline{\mathrm{R}\cap\overline{\mathrm{K}}} \bowtie \mathscr{A}(\rho)$ is a direct consequence of the hypothesis $(\mathrm{K}\cup\{\mathtt{e}\})[\rho] \dashv \mathrm{I}[\sigma]$. Furthermore, from the same hypothesis and from $\mathtt{nf}(\rho) \xRightarrow{\alpha}$ and $\mathtt{nf}(\sigma) \xRightarrow{\overline{\alpha}}$ it follows that $(\mathrm{K}\cup\{\mathtt{e}\})[\rho_\alpha] \dashv \mathrm{I}[\sigma_\alpha]$, hence we conclude $\mathscr{S}\llbracket \mathtt{nf}(\sigma_\alpha)\rrbracket_{\overline{\mathrm{K}}} \mathrel{\mathscr{R}} \mathtt{dual}(\mathscr{C}\llbracket \mathtt{nf}(\rho_\alpha)\rrbracket_{\mathrm{K}})$ by definition of $\mathscr{R}$.

("if" part) Trivial by the definitions of the encoding, the details are left to the reader. $\square$

By combining Theorem 5 and Proposition 3(3) we derive an interesting dual result, by which $\mathtt{dual}(\mathscr{S}\llbracket \mathrm{I}[\sigma]\rrbracket)$ is the *principal client type* that interacts successfully with $\mathscr{S}\llbracket \mathrm{I}[\sigma]\rrbracket$.

Corollary 1. $\mathrm{K}[\rho] \dashv \mathrm{I}[\sigma]$ *if and only if* $\mathscr{C}\llbracket \mathrm{K}[\rho]\rrbracket \leq \mathtt{dual}(\mathscr{S}\llbracket \mathrm{I}[\sigma]\rrbracket)$.

6 Discussion

The encoding of session types into contracts (Section 4) is simple and almost homomorphic with respect to the operations. The branch is the only operation whose encoding requires some care, by adding extra $\ell.\mathbf{\Omega}$ subterms in the generated contract for those labels ℓ not mentioned in the session type. That is, the encoding renders clearly that processes involved in a session may only perform actions that are explicitly allowed by the type of the partner process. In other words, session types describe communications where at any time exactly one of the two interacting parties has control and no handshaking occurs. Interestingly, by interpreting $\mathbf{\Omega}$ as a catastrophic state of a process and by considering a slightly weaker notion of subcontract relation called *safe must* in [2], we have $\alpha.\mathbf{\Omega} \preceq \mathbf{0}$, meaning that, in practice, action α is *not* guaranteed.

The encoding of contracts into session types (Section 5) manifests an exponential blow up of the encoded contract. This indicates that contracts are more abstract than session types and are capable of expressing more complex synchronization scenarios. Part of this added expressiveness derives from the operators $+$ and $\oplus$, which can be used for composing arbitrary behaviors in a liberal way. As we have anticipated in the introduction, the encoding of $\{a, b\}[a \oplus b]$ explicitly notifies the client if only a is available, or if only b is available, or if both a and b are available. This is possible only if the service has centralized control over its own resources and can decide about the availability of a and b. If, on the other hand, the service is a collection of possibly distributed processes (as in service choreographies) it may be impractical or impossible to provide such information to the client.

The lesson we learn is that there is a strict correspondence between contracts and session types that allows one to use them interchangeably *within the context of dyadic interactions*, where both parties provide centralized control on the communication protocol. Contracts go beyond session types in that they permit to characterize *arbitrary processes* in addition to *sessions*. For instance, the process $\overline{a} \mid \overline{b}$, which stands for the parallel composition of two smaller processes sending messages $\overline{a}$ and $\overline{b}$, can be seen as having two unrelated sessions with type $\oplus\langle a : \mathtt{end}\rangle$ and $\oplus\langle b : \mathtt{end}\rangle$. On the other hand, the *whole* process can be typed according to one of the following behaviors

$$\mathbf{\Omega} \qquad \overline{a}.\mathbf{\Omega} + \overline{b}.\mathbf{\Omega} \qquad \overline{a}.\overline{b} + \overline{b}.\mathbf{\Omega} \qquad \overline{a}.\mathbf{\Omega} + \overline{b}.\overline{a} \qquad \overline{a}.\overline{b} + \overline{b}.\overline{a}$$

representing regular, increasingly accurate approximations of the process behavior.

Recently type systems with sessions have been extended so as to guarantee stronger progress properties [8]. Such type systems must necessarily consider a broader perspective that takes into account the mutual dependencies and interactions between sessions. In this respect, we plan to investigate whether the additional expressiveness of contracts already provides the required machinery for dealing with these issues.

References

1. A. Asperti and C. Laneve. Interaction systems I: The theory of optimal reductions. *Mathematical Structures in Computer Science*, 4(4):457–504, 1994.
2. M. Boreale, R. De Nicola, and R. Pugliese. Basic observables for processes. *Information and Computation*, 149(1):77–98, 1999.
3. S. Carpineti, G. Castagna, C. Laneve, and L. Padovani. A formal account of contracts for Web Services. In *WS-FM'06, 3rd Int. Workshop on Web Services and Formal Methods*, number 4184 in LNCS, pages 148–162. Springer, 2006.
4. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for Web Services. In *POPL'08: Proceedings of 35th ACM SIGPLAN SIGACT Symposium on Principles of Programming Languages*. ACM, 2008.
5. R. De Nicola. Two complete axiom systems for a theory of communicating sequential processes. *Information and Control*, 64(1–3):136–172, 1985.
6. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
7. R. De Nicola and M. Hennessy. CCS without τ's. In *TAPSOFT'87/CAAP'87*, number 249 in LNCS, pages 138–152. Springer, 1987.
8. M. Dezani-Ciancaglini, U. de' Liguoro, and N. Yoshida. On Progress for Structured Communications. In *TGC'07*, LNCS. Springer, 2007. To appear.
9. M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, and S. Drossopoulou. Session Types for Object-Oriented Languages. In *ECOOP'06*, volume 4067 of *LNCS*, pages 328–352. Springer, 2006.
10. S. Gay and M. Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2-3):191–225, 2005.
11. M. Hennessy. Acceptance trees. *JACM: Journal of the ACM*, 32(4):896–928, 1985.
12. M. Hennessy. *Algebraic Theory of Processes.* Foundation of Computing. MIT Press, 1988.
13. K. Honda. Types for dyadic interaction. In *CONCUR'93*, number 715 in LNCS, pages 509–523, 1993.

14. C. Laneve and L. Padovani. The *must* preorder revisited – an algebraic theory for web services contracts. In *CONCUR'07, 18th Int. Conference on Concurrency Theory*, number 4703 in LNCS, pages 212–225. Springer, 2007. Full version available at `http://www.sti.uniurb.it/padovani/Papers/lncs_4703_full.pdf`.
15. V. Vasconcelos, S. Gay, and A. Ravara. Type checking a multithreaded functional language with session types. *TCS: Theoretical Computer Science*, 368, 2006.