Using higher-order contracts to model session types* (Extended Abstract)

Giovanni Bernardi[1] and Matthew Hennessy[2]

[1] IMDEA Software Institute, Madrid
bernargi@tcd.ie
[2] School of Computer Science and Statistics, University of Dublin, Trinity College
matthew.hennessy@cs.tcd.ie

Abstract. Session types are used to describe and structure interactions between independent processes in distributed systems. Higher-order types are needed in order to properly structure delegation of responsibility between processes. In this paper we show that higher-order web-service contracts can be used to provide a fully-abstract model of recursive higher-order session types. The model is set-theoretic, in the sense that the denotation of a contract is given by the set of contracts with which it complies; we use a novel notion of *peer* compliance. A crucial step in the proof of full-abstraction is showing that every contract has a non-empty denotation.

1 Introduction

The purpose of this paper is to show that recursive higher-order session types [15], [11] can be given a behavioural interpretation using web-service contracts [19], which is fully-abstract with respect to the Gay & Hole subtyping [13]. Higher-order session types are necessary to handle *session delegation*, and in turn this calls for the development of a novel form of *peer compliance* between higher-order contracts. Our model interprets a higher-order session type as the set of session types, again higher-order, with which it *complies*. This is formalised by viewing session types as *contracts* [19] and using a notion of compliance, which we call *peer* compliance. The completeness of the model relies on showing that every type has at least one other type with which it complies. We prove this using the recently suggested *type complement* [6]. We also believe that this type complement captures the intuition of complementary behaviour more faithfully than the standard notion of type duality from [15]; in the full report [5] we show that type-checking systems for session types, such as in [23], can be improved by using *type complement* rather than *type duality*.

Session types: The interactions between processes in a complex distributed system often follow a pre-ordained pattern. Session types [21,15] have been proposed as a mechanism for concisely describing and structuring these interactions. As a simple example consider a system consisting of two entities

$$(\nu s)\,(\texttt{urls}?(x^+ : S).\mathbf{store} \parallel \texttt{urls}![\,s^+\,].\mathbf{cstmr})$$

* Research supported by SFI project SFI 06 IN.1 1898, and FCT project PTDC/EIA-CCO/122547/2010.

which first exchange a new private communication channel or *session*, s, over the public address of the store `urls`; using the conventions of [13] the customer sends to the store one endpoint of this private session, namely s^+, and keeps the other endpoint s^- for itself. The session type S determines the nature of the subsequent interaction allowed between the two entities; as an example S could be

$$?[\,\texttt{Id}\,];\&\langle\, \texttt{l}_1\colon\ ![\,\texttt{Addr}\,];?[\,\texttt{Int}\,];T, \texttt{l}_2\colon\ ![\,\texttt{Addr}\,];?[\,\texttt{Int}\,];T\,\rangle \tag{1}$$

where `Int`, `Addr`, `Id` are some base types of integers, addresses and credentials respectively, and $\&\langle\, \texttt{l}_1\colon S_1, \ldots, \texttt{l}_\texttt{n}\colon S_n\,\rangle$ is a *branch* type which accepts a choice between interaction on any of the predefined labels $\texttt{l}_\texttt{i}$, followed by the interaction described by the residual type S_i. Thus (1) above dictates that **store** offers a sequence of four interactions on its end s^+ of the session, namely (i) reception of credential, (ii) acceptance of a choice among two commodities labelled by $\texttt{l}_1$, and $\texttt{l}_2$, (iii) followed by the receipt of an address, and (iv) the transmission of a price, of type `Int`; subsequent behaviour is determined by the type T.

The behaviour of **cstmr** on the other end of the session, s^-, is required to match the behaviour described by S, thus satisfying a session type which is intuitively dual to S. For example, the dual to (1) above is

$$![\,\texttt{Id}\,];\oplus\langle\, \texttt{l}_1\colon\ ?[\,\texttt{Addr}\,];![\,\texttt{Int}\,];T', \texttt{l}_2\colon\ ?[\,\texttt{Addr}\,];![\,\texttt{Int}\,];T'\,\rangle \tag{2}$$

under the assumption that T' is the dual of T. Intuitively, input is dual to output and the dual to a branch type is a *choice* type $\oplus\langle\, \texttt{l}_1\colon S_1, \ldots\ \texttt{l}_\texttt{k}\colon S_k\,\rangle$, which allows the process executing the role described by the type to choose one among the labels $\texttt{l}_\texttt{i}$. These two principles lead to a general definition of the *dual* of a session type T, denoted $\overline{T}$ in [21,15].

In order to allow flexibility to the processes fulfilling the roles described by these types a subtyping relation between session types, $T \preccurlyeq S$, is essential; see [13] for a description of the crucial role played by subtyping. Intuitively $T \preccurlyeq S$ means that any process or component fulfilling the role dictated by the session type S may be used where one is required to fulfil the role dictated by T. Thus subtyping gives an intuitive *comparative semantics* to session types. In Definition 1 of Section 2 we slightly generalise the standard definition of [13], so as to account also for base types such as `Int` and `Bool`.

Recursive types are necessary in order to handle sessions which may allow interactions between their endpoints to go on indefinitely.

Example 1. [An ever-lasting session]

$$\begin{aligned} D_s &= X(y) := y \triangleright \{\, \texttt{plus}\colon y?(x)\ \texttt{in}\ y?(z)\ \texttt{in}\ y![\,x+z\,].X[\,y\,]\ \|\\ &\qquad\qquad \texttt{pos}\colon y?(z)\ \texttt{in}\ x![\,z>0\,].X[\,y\,]\,\}\\ D_c &= Y(x) := x \triangleleft \{\, \texttt{pos}\colon x![\,\mathit{random}()\,].x?(z)\ \texttt{in}\ Y[\,x\,]\,\}\\ P &= (\nu\kappa)\,(\texttt{def}\ D_s\ \texttt{in}\ \texttt{def}\ D_c\ \texttt{in}\ X[\,\kappa^+\,] \parallel Y[\,\kappa^-\,]) \end{aligned}$$

The peer $X[\,\kappa^+\,]$, defined by instantiating D_s, accepts over κ^+ the invocation of one of the two methods `plus` and `pos`, reads the actual parameters, sends the result of the chosen

method and starts again. The peer $Y[\,\kappa^-\,]$, defined by instantiating D_c, invokes via its endpoint κ^- the method `pos`, sends a random number, reads the result of the invoked method, and also starts again. The composition of these two peers in P results in a never ending session in which interaction occurs between the two peers forever. Note that the definition of D_s is a recursive version of the math server of [13]. □

The type T that describes the behaviour of D_s on an endpoint k^+ is naturally expressed using recursion:

$$\mu X.\&\langle\, \texttt{plus}\colon\ ?[\,\texttt{Int}\,];?[\,\texttt{Int}\,];![\,\texttt{Int}\,];X, \texttt{pos}\colon\ ?[\,\texttt{Int}\,];![\,\texttt{Int}\,];X\,\rangle$$

Contracts: Web services [19,9] are distributed components which may be combined and extended to offer services to clients. These services are advertised using *contracts*, which are high-level descriptions of the expected behaviour of services. These contracts come equipped with a *sub-contract* relation $\texttt{cnt}_1 \sqsubseteq \texttt{cnt}_2$; intuitively this means that the contract $\texttt{cnt}_2$, or rather a service offering the behaviour described by this contract, may be used as a service which is required to provide the contract $\texttt{cnt}_1$; these abstract contracts are reminiscent of process calculi as CCS and CSP [18,14].

Contracts are very similar, at least syntactically, to sessions types; for example (2) above can very easily be read as the following process description from CCS, $!(\texttt{Id}).(?\texttt{l}_1.?\texttt{Addr}.!\texttt{Int}.\texttt{cnt}' + ?\texttt{l}_2.?\texttt{Addr}.!\texttt{Int}.\texttt{cnt}')$. In fact in Section 3 we give the obvious translation $\mathcal{M}$ from the language of session types to that of contracts; however we continue to use the two distinct languages in order to emphasise the intended use of terms. Then if we provide a behavioural theory of contracts it should be possible to explain how session types determine process behaviour via this mapping $\mathcal{M}$, at least along individual sessions. Indeed steps in this direction have already been made in [1,4] restricting session types to the first-order ones, that is types that cannot express session delegation. But, as we will now explain, the use of delegation in session types requires the use of higher-order types, and in turn higher-order contracts, for which suitable behavioural theories are lacking.

Session delegation: Consider the following system where the customer **cstmr** is replaced by **girlf** and there are now four components:

$$\begin{aligned}(\nu s)\,(\nu p)\,(\nu b)\,(&\texttt{urls}![\,s^+\,].\texttt{urlb}![\,p^+\,].\texttt{urlb}![\,b^+\,].\mathbf{girlf} \parallel\\ &\texttt{urls}?(s^+\colon S).\mathbf{store} \parallel \texttt{urls}?(p^+\colon S_p).\mathbf{bank} \parallel\\ &\texttt{urlbf}?(b^+\colon S_b).\mathbf{boyf})\end{aligned}$$

Three private sessions s, p, b are created and the positive endpoints are distributed to the **store**, **bank**, and **boyf** respectively. One possible script for the new customer **girlf** is as follows:

(i) send credential to **store**: send id on session s^-
(ii) **delegate** choice of commodity to **boyf**: send session b^- on session s^-
(iii) await **delegation** from **boyf** to arrange payment: receive session s^- back on session b^-.

Thus the session type S_b at which the boyfriend uses the session end b^+ must countenance both the reception and transmission of session ends, rather than simply data. In

this case we can take S_b to be the higher-order session type $?[\,T_1\,];![\,T_2\,];\text{END}$, where in turn T_1 is the session type $\oplus\langle\, \mathtt{l}_1 :\ ?[\,\mathtt{Addr}\,];\text{END}\,\rangle$ and T_2 must allow **girlf** to arrange payment through the **bank**. This in turn means that T_2 is a higher-order session type as payment will involve the transmission of the payment session p.

The combination of delegation and recursion leads to processes with complicated behaviour which in turn puts further strain on the system of session types.

Example 2. [Everlasting generation of finite sessions]
We use the syntax of [23]. Consider the process $P = (\nu\kappa_{\mathbf{o}})\,(\mathtt{def}\ D\ \mathtt{in}\ X[\,\kappa_{\mathbf{o}}^{+},\,\kappa_{\mathbf{o}}^{-}\,])$, where $D := X(x,y) = (\nu\kappa_{\mathbf{f}})\,(\mathtt{throw}\ x[\,\kappa_{\mathbf{f}}^{+}\,];\mathbf{0} \parallel \mathtt{catch}\ y(z)\ \mathtt{in}\, X[\,z,\,\kappa_{\mathbf{f}}^{-}\,])$. Intuitively, at each iteration the code $X[\,\kappa_{\mathbf{o}}^{+},\kappa_{\mathbf{f}}^{-}\,]$ has the two endpoints of a pre-existing session, $\kappa_{\mathbf{o}}$, delegates over the endpoint $\kappa_{\mathbf{o}}^{+}$ the endpoint $\kappa_{\mathbf{f}}^{+}$, and then recursively repeats the loop using $\kappa_{\mathbf{f}}$ as pre-existing session.

According to the reduction semantics in [23] the execution of P will never give rise to a communication error or a deadlock. But the endpoint $\kappa_{\mathbf{f}}^{+}$ can only be assigned a session type of the form $\mu X.\,![\,X\,];\text{END}$. Such types are forbidden in [2] but they are allowed in the typing systems of [15,13,23,22]. □

If session types are to be explained behaviourally via the translation $\mathcal{M}$ into contracts, the target language of contracts needs to be higher-order. For instance, the type $?[\,T_1\,];![\,T_2\,];\text{END}$ is mapped by $\mathcal{M}$ to the contract $?(\,!\mathtt{l}_1.?(\mathtt{Addr}).\mathtt{1}\,).?(\,\mathtt{cnt}_2\,).\mathtt{1}$, where $\mathtt{cnt}_2 = \mathcal{M}(T_2)$. This in turn means that we require a behavioural theory of higher-order contracts. This is the topic of the current paper. In particular we develop a novel sub-contract preorder, which we refer to as the *peer* sub-contract preorder $\sqsubseteq\!\!\!\approx$ with the property that, for all session types,

$$S \preccurlyeq T \text{ if and only if } \mathcal{M}(S) \sqsubseteq\!\!\!\approx \mathcal{M}(T) \tag{3}$$

On the left hand-side we have the subtyping preorder between session types, which determines when processes with session type T can play the role required by type S; on the right-hand side we have a behaviourally determined sub-contract preorder between the interpretation of the types as higher-order contracts. This behavioural preorder is defined in terms of a novel definition of *peer compliance* between these contracts.

In the remainder of this Introduction we briefly outline how the *peer* sub-contract preorder is defined. Intuitively $\sigma_1 \sqsubseteq\!\!\!\approx \sigma_2$, where σ_i are contracts, if every contract ρ which *complies* with σ_1 also *complies* with σ_2. In turn the intuition behind *compliance* is as follows. We say that a contract ρ complies with contract σ, written $\rho \dashv_{r2r} \sigma$, if any pair of processes in the source language p, q which guarantee the contracts ρ, σ respectively, can interact indefinitely to their mutual satisfaction; in particular if no further interaction is possible between them, individually they both have reached *successful* or *happy* states. We call this concept *mutual* or *peer* compliance, as both participants are required to attain a *happy* state simultaneously. This is in contrast to [9,19,4] where an asymmetric compliance is used, in which only one participant, the client, is required to reach a *happy* state.

In this paper, rather than discussing processes in the source language, how they can interact and how they guarantee contracts, we mimic the interaction between processes

using a symbolic semantics between contracts. We define judgements of the form

$$\rho \parallel \sigma \xrightarrow{\tau} \rho' \parallel \sigma' \tag{4}$$

meaning that if p, q, from the source language, guarantee the contracts ρ, σ respectively, then they can interact and evolve to processes p', q' which guarantee the residual contracts ρ', σ' respectively.

For example we will have the judgement $!\texttt{Int}.\rho' \parallel ?\texttt{Real}.\sigma' \xrightarrow{\tau} \rho' \parallel \sigma'$. On the left-hand side of the parallel constructor $\parallel$ we have a contract guaranteed by a process that supplies an `Int`; on the right-hand side there is a contract guaranteed by a process which will accept a datum that can be used as a real. Since we are assuming that integers can be interpreted as reals, that is $\texttt{Int} \leqslant_{\mathsf{b}} \texttt{Real}$, we know that an interaction described by the judgement above takes place.

However it is unclear when an interaction of the form

$$!(\sigma_1).\rho' \parallel ?(\sigma_2).\sigma' \xrightarrow{\tau} \rho' \parallel \sigma' \tag{5}$$

should take place. Here on the left is a contract satisfied by a process which provides a session endpoint that satisfies the contract σ_1; on the right is a contract satisfied by a process that accepts any session endpoint which guarantees the contract σ_2. Intuitively the interaction should be allowed if σ_1 is a sub-contract of σ_2, that is $\sigma_2 \precsim \sigma$. However the whole purpose of defining the judgements (4) above is in order to define the preorder $\precsim$; there is a circularity in our arguments.

We break this circularity by supposing a predefined sub-contract preorder $\mathcal{B}$ and allowing the interaction (5) whenever $\sigma_1 \ \mathcal{B} \ \sigma_2$. More generally we develop a parametrised theory, with interaction judgements of the form $\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma'$ leading to a parametrised peer-compliance relation $\sigma \dashv^{\mathcal{B}}_{\text{p2p}} \rho$ which in turn leads to a parametrised sub-contract preorder $\rho_1 \sqsubseteq^{\mathcal{B}} \rho_2$. We then prove the main result of the paper, (3) above, by showing:

> There exists some preorder $\mathcal{B}_0$ over higher-order contracts such that $S \preccurlyeq T$ if and only if $\mathcal{M}(S) \sqsubseteq^{\mathcal{B}_0} \mathcal{M}(T)$

This particular preorder $\mathcal{B}_0$, which we construct and in (3) above has been referred to as $\precsim$, has a natural behavioural interpretation. It satisfies the behavioural equation

$$\sigma_1 \ \mathcal{B}_0 \ \sigma_2 \text{ if and only if } \sigma_1 \sqsubseteq^{\mathcal{B}_0} \sigma_2 \tag{6}$$

Moreover it is the largest preorder between higher-order contracts which satisfies (6).

The proof of (6) depends on an alternative syntactic characterisation of the set-based preorders $\sqsubseteq^{\mathcal{B}}$ which in turn relies crucially on a natural property of the peer-compliance relations:

> For every contract σ there exists a complementary contract, $\mathsf{cplmt}(\sigma)$, which complies with it, $\sigma \dashv^{\mathcal{B}}_{\text{p2p}} \mathsf{cplmt}(\sigma)$. $(\star)$

In view of the natural correspondence between contracts and session types there is a natural candidate for complementary contracts. Intuitively the dual of a type $\overline{T}$ is designed to capture the complementary behaviour expressed by the type T. Moreover the

duality function on session types discussed on page 2 immediately extends to contracts; specifically we can define $\overline{\sigma}$ to be $\mathcal{M}(\overline{\mathcal{M}^{-1}(\sigma)})$.

However, somewhat surprisingly, there are contracts σ which do not comply with their duals, $\sigma \not\dashv^{\mathcal{B}}_{\text{r2p}} \overline{\sigma}$; see Example 4. However ($\star$) above can be established by using instead a different notion of dual, first proposed in [6] for typing *copyless message-passing processes*. We also believe that this alternative notion, which in this paper we call *complement*, captures the intuitive notion of complementary behaviour more faithfully than the standard duality.

Paper structure: In Section 2 we recall the standard theory of recursive higher-order session types, while Section 3 introduces higher-order contracts and our novel parametrised peer sub-contract preorder $\sqsubseteq^{\mathcal{B}}$. Although the definition of this preorder is set-theoretic, it can be characterised using only the syntactic form of contracts; this stems from the very restricted form that our higher-order contracts can take. This is also discussed in Section 3. Using this syntactic characterisation we develop enough properties of the preorders $\sqsubseteq^{\mathcal{B}}$ to ensure the existence of the particular preorder $\mathcal{B}_0$ alluded to in (6) above; this is the topic of Section 4. The complementation operator on contracts from [6], $\mathsf{cplmt}(\sigma)$, alluded to above is also defined and discussed in Section 4. Related work is then discussed in Section 5.

All the proofs and the technical details are omitted from this extended abstract, and can be found in the companion report [5].

2 Session types

Here we recall, using the notation from [13], the standard theory of subtyping for recursive session types. The grammar for the language L_{STyp} of session type terms is given by the following grammar, which uses a collection of unspecified base types $\mathtt{BT}$, of which we enumerate a sample.

$$\begin{aligned} S, T &::= \text{END} \mid X \mid ?[\,M\,]; S \mid ![\,M\,]; S \mid \mu X.S \mid \&\langle\, \mathtt{l}_1 \colon S_1, \ldots, \mathtt{l}_n \colon S_n \,\rangle \\ &\quad \oplus\langle\, \mathtt{l}_1 \colon S_1, \ldots, \mathtt{l}_n \colon S_n \,\rangle \\ M, N &::= S \mid \mathtt{t} \\ \mathtt{t} &::= \mathtt{Id}, \mathtt{Addr}, \mathtt{Int}, \mathtt{Real}, \ldots \end{aligned}$$

In the grammar above we assume $n \geq 1$; moreover we use a denumerable set of labels, $\mathtt{L} = \{\mathtt{l}_1, \mathtt{l}_2, \mathtt{l}_3, \ldots\}$, in the *branch* and *choice* constructs. Recall from the Introduction that $\&\langle\, \mathtt{l}_1 \colon S_1, \ldots, \mathtt{l}_n \colon S_n \,\rangle$ offers different possible behaviours based on a set of labels $\{\mathtt{l}_1, \mathtt{l}_2, \mathtt{l}_3, \ldots \mathtt{l}_n\}$ while $\oplus\langle\, \mathtt{l}_1 \colon S_1, \ldots, \mathtt{l}_n \colon S_n \,\rangle$ takes a choice of behaviours; in both constructs the labels used are assumed to be distinct.

We use STyp to denote the set of session type terms in L_{STyp} which are *closed* and *guarded*; both these concepts have standard definitions, which may be found in [5, Appendix A]. We refer to the terms in STyp as session types. For instance $\mu X.X$ and $\&\langle\, \mathtt{tea} \colon \mu X.X \,\rangle$ are not in STyp.

Subtyping is defined coinductively and uses some unspecified subtyping preorder $\leqslant_{\mathtt{b}}$ between base types, a typical example being $\mathtt{Int} \leqslant_{\mathtt{b}} \mathtt{Real}$, meaning that an integer may be supplied where a real number is required. Recursive types are handled by a

standard function $\mathsf{unfold}(T)$ which unfolds all the first-level occurrences of $\mu X.-$ in the (guarded) type T. The formal definition of unfold in turn depends on the definition of substitution $T\{S/X\}$, the syntactic substitution of the term S for all free occurrences of X in T. The details may be found in [5].

Definition 1. *[Subtyping]*
Let $\mathcal{F}\leqslant : \mathcal{P}(\mathsf{STyp}^2) \longrightarrow \mathcal{P}(\mathsf{STyp}^2)$ *be the functional defined so that* $(T,U) \in \mathcal{F}\leqslant(\mathcal{R})$ *whenever one of the following holds:*

(i) *if* $\mathsf{unfold}(T) = \text{END}$ *then* $\mathsf{unfold}(U) = \text{END}$
(ii) *if* $\mathsf{unfold}(T) = ?[\,\mathtt{t}_1\,];S_1$ *then* $\mathsf{unfold}(U) = ?[\,\mathtt{t}_2\,];S_2$ *and* $S_1\ \mathcal{R}\ S_2$ *and* $\mathtt{t}_1 \leqslant_{\mathtt{b}} \mathtt{t}_2$
(iii) *if* $\mathsf{unfold}(T) = ![\,\mathtt{t}_1\,];S_1$ *then* $\mathsf{unfold}(U) = ![\,\mathtt{t}_2\,];S_2$ *and* $S_1\ \mathcal{R}\ S_2$ *and* $\mathtt{t}_2 \leqslant_{\mathtt{b}} \mathtt{t}_1$
(iv) *if* $\mathsf{unfold}(T) = ![\,T_1\,];S_1$ *then* $\mathsf{unfold}(U) = ![\,T_2\,];S_2$ *and* $S_1\ \mathcal{R}\ S_2$ *and* $T_2\ \mathcal{R}\ T_1$
(v) *if* $\mathsf{unfold}(T) = ?[\,T_1\,];S_1$ *then* $\mathsf{unfold}(U) = ?[\,T_2\,];S_2$ *and* $S_1\ \mathcal{R}\ S_2$ *and* $T_1\ \mathcal{R}\ T_2$
(vi) *if* $\mathsf{unfold}(T) = \&\langle\,\mathtt{l}_1\colon S_1, \ldots l_m\colon S_m\,\rangle$ *then* $\mathsf{unfold}(U) = \&\langle\,\mathtt{l}_1\colon S'_1, \ldots, \mathtt{l}_n\colon S'_n\,\rangle$ *where* $m \leq n$ *and* $S_i\ \mathcal{R}\ S'_i$ *for all* $i \in [1,\ldots,m]$
(vii) *if* $\mathsf{unfold}(T) = \oplus\langle\,\mathtt{l}_1\colon S_1, \ldots l_m\colon S_m\,\rangle$ *then* $\mathsf{unfold}(U) = \oplus\langle\,\mathtt{l}_1\colon S'_1, \ldots, \mathtt{l}_n\colon S'_n\,\rangle$ *where* $n \leq m$ *and* $S_i\ \mathcal{R}\ S'_i$ *for all* $i \in [1,\ldots,n]$

If $\mathcal{R} \subseteq \mathcal{F}\leqslant(\mathcal{R})$*, then we say that* R *is a* type simulation*. Standard arguments ensure that there exists the greatest solution of the equation* $\mathcal{R} = \mathcal{F}\leqslant(\mathcal{R})$*; we call this solution the* subtyping*, and we denote it* $\leqslant$*.* □

Intuitively $S \leqslant T$ means that processes adhering to the role dictated by T may be used where processes following the role dictated by S are required. Our aim is to formalise this intuition by proving that the higher-order contracts determined by these types, respectively $\mathcal{M}(S)$ and $\mathcal{M}(T)$ are related behaviourally, using our notion of *peer* compliance.

3 Higher-order contracts

Here first we define higher-order session contracts and explain the set-based subcontract preorder on them; this uses the notion of *peer compliance* between them. Afterwards we characterise up-to a parameter $\mathcal{B}$ this set-based preorder. We do so by comparing the purely syntactic structure of contracts.

The grammar for the language of contract terms L_{SCts} is:

$$\rho,\sigma ::= \mathsf{1} \mid ?\mathtt{t}.\sigma \mid !\mathtt{t}.\sigma \mid !(\sigma).\sigma \mid ?(\sigma).\sigma \mid x \mid \mu x.\sigma \mid \sum_{i\in I} ?\mathtt{l}_\mathtt{i}.\sigma_i \mid \bigoplus_{i\in I} !\mathtt{l}_\mathtt{i}.\sigma_i$$

where we assume the labels $\mathtt{l}_i$s to be pairwise distinct and the set I to be non-empty. We use SCts to denote the set of terms which are guarded and closed. These will be referred to as higher-order session contracts, or simply contracts. When I is a singleton set $\{k\}$, we write $!\mathtt{l}_\mathtt{k}.\sigma_k$ and $?\mathtt{l}_\mathtt{k}.\sigma_k$ in place of $\bigoplus_{i\in I}!\mathtt{l}_\mathtt{i}.\sigma_i$ and $\sum_{i\in I}?\mathtt{l}_\mathtt{i}.\sigma_i$.

The operational meaning of contracts is given by interpreting them as processes from a simple process calculus. To this end let Act, ranged over by λ, be the union of three sets, namely $\{\,?\mathtt{l},\ !\mathtt{l} \mid \mathtt{l} \in \mathtt{L}\,\}$, $\{\,?\mathtt{t},\ !\mathtt{t} \mid \mathtt{t} \in \mathtt{BT}\,\}$, and $\{\,?(\sigma),\ !(\sigma) \mid \sigma \in \mathsf{SCts}\,\}$.

We use Act_τ to denote the set $\mathsf{Act} \cup \{\tau\}$ to emphasise that the special symbol τ is not in Act. We define judgements of the form $\sigma_1 \xrightarrow{\mu} \sigma_2$, where $\mu \in \mathsf{Act}_\tau$ and $\sigma_1, \sigma_2 \in \mathsf{SCts}$, by using the following (standard) axioms, where $|\, I \,|$ is the cardinality of I,

$$\frac{}{\lambda.\sigma \xrightarrow{\lambda} \sigma}\ \lambda \in \mathsf{Act} \qquad \frac{}{\mu x.\sigma \xrightarrow{\tau} \sigma\{\mu x.\sigma/x\}}$$

$$\frac{}{\bigoplus_{i\in I} !\mathtt{l}_\mathtt{i}.\sigma_i \xrightarrow{\tau} !\mathtt{l}_\mathtt{i}.\sigma_i}\ |\, I \,| > 1 \qquad \frac{}{\sum_{i\in I} ?\mathtt{l}_\mathtt{i}.\sigma_i \xrightarrow{?\mathtt{l}_\mathtt{i}} \sigma_i}$$

We also have the special judgement $\mathsf{1} \xrightarrow{\checkmark}$, which formalises operationally that $\mathsf{1}$ is the *satisfied* contract. Although terms like $!\mathtt{l}.\sigma$ stand actually for singleton internal sums, we infer their semantics by using the rule for prefixes; for example $!\mathtt{l}.\sigma \xrightarrow{!\mathtt{l}} \sigma$.

In order to define the *peer compliance* between two contracts ρ, σ, we also need to say when two processes p, q satisfying these contracts can interact. This is formalised indirectly as a relation of the form $\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma'$ which, as explained in the Introduction, is designed to capture the intuition that if processes p, q satisfy the contracts ρ, σ respectively, then they can interact and their residuals will satisfy the residual contracts ρ', σ' respectively.

The relation $\longrightarrow_{\mathcal{B}}$ is determined by the following inference rules:

$$\frac{\rho \xrightarrow{\tau} \rho'}{\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma} \qquad \frac{\sigma \xrightarrow{\tau} \sigma'}{\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho \parallel \sigma'} \qquad \frac{\rho \xrightarrow{\lambda_1} \rho' \quad \sigma \xrightarrow{\lambda_2} \sigma'}{\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma'}\ \lambda_1 \bowtie_{\mathcal{B}} \lambda_2$$

This reduction relation is parametrised on a relation $\sigma_1\ \mathcal{B}\ \sigma_2$ between contracts, which determines when the contract σ_1 can be accepted when σ_2 is required. Using such a $\mathcal{B}$ we define an *interaction* relation between contracts as follows:

$$\lambda_1 \bowtie_{\mathcal{B}} \lambda_2 = \begin{cases} \lambda_1 = !\mathtt{l},\ \lambda_2 = ?\mathtt{l} & \\ \lambda_1 = ?\mathtt{l},\ \lambda_2 = !\mathtt{l} & \\ \lambda_1 = !\mathtt{t}_1,\ \lambda_2 = ?\mathtt{t}_2 & \mathtt{t}_1 \leqslant_\mathtt{b} \mathtt{t}_2 \\ \lambda_1 = ?\mathtt{t}_1,\ \lambda_2 = !\mathtt{t}_2 & \mathtt{t}_2 \leqslant_\mathtt{b} \mathtt{t}_1 \\ \lambda_1 = !(\sigma_1),\ \lambda_2 = ?(\sigma_2) & \sigma_1\ \mathcal{B}\ \sigma_2 \\ \lambda_1 = ?(\sigma_1),\ \lambda_2 = !(\sigma_2) & \sigma_2\ \mathcal{B}\ \sigma_1 \end{cases}$$

Essentially the relation $\bowtie_{\mathcal{B}}$ treats $\mathcal{B}$ as a subtyping on contracts.

Definition 2. *[$\mathcal{B}$-Peer compliance]*
Let $C^{\mathrm{r2p}} : \mathcal{P}(\mathsf{SCts}^2) \times \mathcal{P}(\mathsf{SCts}^2) \longrightarrow \mathcal{P}(\mathsf{SCts}^2)$ *be the rule functional defined so that* $(\rho, \sigma) \in C^{\mathrm{r2p}}(\mathcal{R}, \mathcal{B})$ *whenever both the following conditions hold:*

(i) if $\rho \parallel \sigma \not\xrightarrow{\tau}_{\mathcal{B}}$ *then* $\rho \xrightarrow{\checkmark}$ *and* $\sigma \xrightarrow{\checkmark}$
(ii) if $\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma'$ *then* $\rho'\ \mathcal{R}\ \sigma'$

If $\mathcal{R} \subseteq C^{\mathrm{r2p}}(\mathcal{R}, \mathcal{B})$*, then we say that* $\mathcal{R}$ *is a* $\mathcal{B}$-coinductive peer compliance*. Fix a* $\mathcal{B}$*. Standard arguments ensure that there exists the greatest solution of the equation* $X = C^{\mathrm{r2p}}(X, \mathcal{B})$*; we call this solution the* $\mathcal{B}$-peer compliance*, and we denote it* $\dashv^{\mathcal{B}}_{\mathrm{r2p}}$*.* □

The intuition here is that if $\rho \dashv^{\mathcal{B}}_{\text{p2p}} \sigma$ then processes satisfying these contracts can interact safely; the co-inductive nature of the definition even allows this interaction to continue forever. But if a point is reached where no further interaction is allowed condition (i) means that both participants must be *happy* simultaneously; that is they must be able to perform the success action $\checkmark$.

Definition 3. *[$\mathcal{B}$-peer subcontract preorder]*
For $\sigma_1, \sigma_2 \in \mathsf{SCts}$ *let* $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ *whenever* $\rho \dashv^{\mathcal{B}}_{\text{p2p}} \sigma_1$ *implies* $\rho \dashv^{\mathcal{B}}_{\text{p2p}} \sigma_2$, *for every* $\rho \in \mathsf{SCts}$. □

The parametrised peer subcontract preorder $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ is set based, and quantifies over all peers in $\mathcal{B}$-compliance with σ_1. However, because of the restricted syntax of higher-order contracts, it turns out that $\sqsubseteq^{\mathcal{B}}$ can be characterised by the syntactic structure of σ_1 and σ_2, at least for behavioural preorders $\mathcal{B}$ which satisfy certain minimal conditions.

Definition 4. *[$\mathcal{B}$-syntactic peer preorder]*
Let $\mathcal{S} : \mathcal{P}(\mathsf{SCts}^2) \times \mathcal{P}(\mathsf{SCts}^2) \longrightarrow \mathcal{P}(\mathsf{SCts}^2)$ *be the functional defined so that* $(\sigma_1, \sigma_2) \in \mathcal{S}(\mathcal{R}, \mathcal{B})$ *whenever one of the following holds:*

(i) *if* $\mathsf{unfold}(\sigma_1) = 1$ *then* $\mathsf{unfold}(\sigma_2) = 1$
(ii) *if* $\mathsf{unfold}(\sigma_1) = ?\mathtt{t}_1.\sigma_1'$ *then* $\mathsf{unfold}(\sigma_2) = ?\mathtt{t}_2.\sigma_2'$ *and* $\sigma_1' \,\mathcal{R}\, \sigma_2'$ *and* $\mathtt{t}_1 \preccurlyeq_{\mathtt{b}} \mathtt{t}_2$
(iii) *if* $\mathsf{unfold}(\sigma_1) = !\mathtt{t}_1.\sigma_1'$ *then* $\mathsf{unfold}(\sigma_2) = !\mathtt{t}_2.\sigma_2'$ *and* $\sigma_1' \,\mathcal{R}\, \sigma_2'$ *and* $\mathtt{t}_2 \preccurlyeq_{\mathtt{b}} \mathtt{t}_1$
(iv) *if* $\mathsf{unfold}(\sigma_1) = !(\sigma_1'').\sigma_1'$ *then* $\mathsf{unfold}(\sigma_2) = !(\sigma_2'').\sigma_2'$ *and* $\sigma_1' \,\mathcal{R}\, \sigma_2'$ *and* $\sigma_2'' \,\mathcal{B}\, \sigma_1''$
(v) *if* $\mathsf{unfold}(\sigma_1) = ?(\sigma_1'').\sigma_1'$ *then* $\mathsf{unfold}(\sigma_2) = ?(\sigma_2'').\sigma_2'$ *and* $\sigma_1' \,\mathcal{R}\, \sigma_2'$ *and* $\sigma_1'' \,\mathcal{B}\, \sigma_2''$
(vi) *if* $\mathsf{unfold}(\sigma_1) = \sum_{i \in I} ?\mathtt{l}_i.\sigma_i^1$ *then* $\mathsf{unfold}(\sigma_2) = \sum_{j \in J} ?\mathtt{l}_j.\sigma_j^2$ *where* $I \subseteq J$ *and* $\sigma_i^1 \,\mathcal{R}\, \sigma_i^2$ *for all* $i \in I$
(vii) *if* $\mathsf{unfold}(\sigma_1) = \bigoplus_{i \in I} !\mathtt{l}_i.\sigma_i^1$ *then* $\mathsf{unfold}(\sigma_2) = \bigoplus_{j \in J} !\mathtt{l}_j.\sigma_j^2$ *where* $J \subseteq I$ *and* $\sigma_j^1 \,\mathcal{R}\, \sigma_j^2$ *for all* $j \in J$

Fix a $\mathcal{B}$. *Since* $\mathcal{S}$ *is monotone ([5, Lemma 3.3]), standard arguments ensure that there exists the greatest solution of the equation* $X = \mathcal{S}(X, \mathcal{B})$; *we call this solution the* $\mathcal{B}$-syntactic peer preorder, *and we denote it by* $\preccurlyeq^{\mathcal{B}}$. □

Our intention is to show that the set-theoretic relation $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ coincides with the more amenable syntactically defined relation $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$, provided $\mathcal{B}$ satisfies some simple properties. In one direction the proof follows directly from the definitions of the relations at issue. In the other we need a non-trivial property of session contracts that we relegate to Section 4; see Theorem 4.

Theorem 1. *Let* $\mathcal{B}$ *be a transitive relation on session contracts. Then* $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$ *implies* $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$.

Example 3. Here we show that Theorem 1 requires the relation $\mathcal{B}$ to be transitive. Let $\mathcal{B}$ be $\{ (1, \sigma), (\sigma, !\mathtt{l}.\,1) \}$, where σ is the contract $!\mathtt{l}.!\mathtt{l}.\,1$; this is obviously not transitive. We show that $\preccurlyeq^{\mathcal{B}} \not\subseteq \sqsubseteq^{\mathcal{B}}$. First, the relation $\mathcal{R} = \{ (\sigma_1, \sigma_2), (1, 1) \}$, where $\sigma_1 = !(\sigma).\,1$, $\sigma_2 = !(1).\,1$, is a prefixed point of $\mathcal{S}$, and thus $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$. Now let $\rho = ?(!\mathtt{l}.\,1).\,1$. The reason why $\sigma_1 \not\sqsubseteq^{\mathcal{B}} \sigma_2$ is that $\rho \dashv^{\mathcal{B}}_{\text{p2p}} \sigma_1$, because $\{ (\rho, \sigma_1), (1, 1) \}$ is a $\mathcal{B}$-coinductive compliance, while $\rho \not\dashv^{\mathcal{B}}_{\text{p2p}} \sigma_2$ because of the computation $\rho \parallel \sigma_2 \stackrel{\tau}{\not\longrightarrow}_{\mathcal{B}}$. □

Theorem 2. *For every preorder on session contracts $\mathcal{B}$, $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ implies $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$.*

Proof (Outline). The argument is by case analysis on $\mathsf{unfold}(\sigma_1)$. For instance, consider the case in which $\mathsf{unfold}(\sigma_1) = !(\mathtt{t}).\sigma_1'$. Thanks to Theorem 4 there exists a ρ' such that $\rho' \dashv^{\mathcal{B}}_{\text{p2p}} \sigma_1'$. It follows that $?(\mathtt{t}).\rho' \dashv^{\mathcal{B}}_{\text{p2p}} \mathsf{unfold}(\sigma_1)$, and so $\rho \dashv^{\mathcal{B}}_{\text{p2p}} \mathsf{unfold}(\sigma_2)$. This is enough to show the properties of $\mathsf{unfold}(\sigma_2)$ required by the definition of $\preccurlyeq^{\mathcal{B}}$.

Corollary 1. *For any preorder $\mathcal{B}$ over session contracts, $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$ if and only if $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$.*

4 Modelling session types

Session types and contracts, formalisms developed independently, are nevertheless just syntactic variations of each other:

$$\begin{array}{llll}
\mathcal{M}(\text{END}) = \mathsf{1}, & \mathcal{M}(X) = x, & \mathcal{M}(\mu X.S) = \mu x.\mathcal{M}(S), & \mathcal{M}(![\,T\,];S') = !(\mathcal{M}(T)).\mathcal{M}(S'), \\
\mathcal{M}(\&\langle\, \mathtt{l}_1 : S_1, \ldots, \mathtt{l}_n : S_n \,\rangle) = \sum_{i\in[1;n]} ?\mathtt{l}_i.\mathcal{M}(S_i), & & & \mathcal{M}(![\,\mathtt{t}\,];S') = !\mathtt{t}.\mathcal{M}(S'), \\
\mathcal{M}(\oplus\langle\, \mathtt{l}_1 : S_1, \ldots, \mathtt{l}_n : S_n \,\rangle) = \bigoplus_{i\in[1;n]} !\mathtt{l}_i.\mathcal{M}(S_i), & & & \mathcal{M}(?[\,T\,];S') = ?(\mathcal{M}(T)).\mathcal{M}(S'), \\
 & & & \mathcal{M}(?[\,\mathtt{t}\,];S') = ?\mathtt{t}.\mathcal{M}(S')
\end{array}$$

Our aim is to show that the subtyping relation between session types, $S \preccurlyeq T$, can be modelled precisely by the set-based contract preorder, $\mathcal{M}(S) \sqsubseteq^{\mathcal{B}} \mathcal{M}(T)$, for a particular choice of $\mathcal{B}$. In order to determine this $\mathcal{B}$ we need to develop some properties of functionals over contracts. Let $\mathcal{P}re$ denote the collection of preorders over the set of contracts SCts; ordered set-theoretically this is a complete lattice [5, Lemma 4.2]. Let $\mathcal{F} : \mathcal{P}re \longrightarrow \mathcal{P}re$ be defined by letting $\mathcal{F}(\mathcal{B})$ be the preorder $\sqsubseteq^{\mathcal{B}}$. By Corollary 1 we know that $\mathcal{F}(\mathcal{B}) = \preccurlyeq^{\mathcal{B}}$, and therefore $\mathcal{F}(\mathcal{B}) = \nu X.\mathcal{S}(X, \mathcal{B})$, from Definition 4. Since $\mathcal{S}$ is monotone in its second parameter ([5, Lemma 3.3 (b)]), the endofunction $\mathcal{F}$ over the complete lattice $\mathcal{P}re$ is monotone. The Knaster-Tarski theorem now ensures that $\mathcal{F}$ has fixed points, in particular a maximal one.

Definition 5. *[Peer subcontract preorder]*
Let $\precsim$ denote $\nu X.\mathcal{F}(X)$, the greatest fixed point of the function $\mathcal{F}$. We refer to $\precsim$ as the Peer subcontract preorder. □

The proof that $\precsim$ provides a fully-abstract model of subtyping $\preccurlyeq$ on session types, relies on a syntactic characterisation of $\precsim$, stated in the next lemma, and it implies a result on the decidability of $\precsim$.

Lemma 1. $\precsim = \nu X.\mathcal{S}(X, X)$.

Theorem 3. *[Full-abstraction]*
For every $T, S \in \mathsf{STyp}$, $S \preccurlyeq T$ if and only if $\mathcal{M}(S) \precsim \mathcal{M}(T)$.

Proof (Outline). The subtyping $\preccurlyeq$ is the greatest fixed point of $\mathcal{F}\preccurlyeq$ by definition, $\precsim$ is the greatest fixed point of $\mathcal{S}$ because of Lemma 1, and $\mathcal{M}$ provides a bijection from prefixed points of $\mathcal{F}\preccurlyeq$ to prefixed points $\mathcal{S}$ ([5, Lemma 4.8, Lemma 4.9]). This is why full-abstraction is true.

Proposition 1. *If $\preccurlyeq_{\mathtt{b}}$ is decidable, then the relation $\sqsubseteq\!\!\!\!\sim$ is decidable.*

Theorem 3 depends on Corollary 1, which depends on Theorem 2. In turn this theorem relies on the existence for every session contract σ of a "complementary" session contract $\mathsf{cplmt}(\sigma)$ that is in $\mathcal{B}$-peer compliance with σ, at least for $\mathcal{B}$s that satisfy certain minimal conditions. To construct $\mathsf{cplmt}(\sigma)$, the well-known *syntactic duality* of session types is an obvious candidate. This is defined inductively as follows [15]:

$$\overline{\text{END}} = \text{END}, \quad \overline{X} = X, \quad \overline{\mu X.S} = \mu X.\overline{S}, \quad \overline{?[\,M\,];S} = ![\,M\,];\overline{S}, \quad \overline{![\,M\,];S} = ?[\,M\,];\overline{S},$$
$$\overline{\&\langle\, \mathtt{l}_1 \colon S_1, \ldots, \mathtt{l}_n \colon S_n\,\rangle} = \oplus\langle\, \mathtt{l}_1 \colon \overline{S_1}, \ldots, \mathtt{l}_n \colon \overline{S_n}\,\rangle,$$
$$\overline{\oplus\langle\, \mathtt{l}_1 \colon S_1, \ldots, \mathtt{l}_n \colon S_n\,\rangle} = \&\langle\, \mathtt{l}_1 \colon \overline{S_1}, \ldots, \mathtt{l}_n \colon \overline{S_n}\,\rangle$$

This operator can also be applied to contracts in the obvious manner, using the injection $\mathcal{M}(-)$.

Example 4. In general it is not true that a contract σ complies with its dual $\overline{\sigma}$. To prove this, we say that the relation $\mathcal{B}$ is *reasonable* whenever $\sigma_1 \mathrel{\mathcal{B}} \sigma_2$ implies the following conditions:

i) $\mathsf{unfold}(\sigma_1) \mathrel{\mathcal{B}} \mathsf{unfold}(\sigma_2)$
ii) $\sigma_1 \xrightarrow{\lambda_1}$ and $\sigma_2 \xrightarrow{\lambda_2}$ imply that λ_1 and λ_2 are *both* input actions or *both* output actions.

If $\mathcal{B}$ is reasonable then we can find a contract σ such that $\sigma \not\dashv^{\mathcal{B}}_{\text{p2p}} \overline{\sigma}$. For example take σ to be $\mu x.?(x).1$; here $\overline{\sigma}$ is $\mu x.!(x).1$. The behaviour of these contracts is $\sigma \xrightarrow{\tau} ?(\sigma).1 \xrightarrow{?(\sigma)} 1 \xrightarrow{\checkmark}$, and $\overline{\sigma} \xrightarrow{\tau} !(\overline{\sigma}).1 \xrightarrow{!(\overline{\sigma})} 1 \xrightarrow{\checkmark}$. If $\mathcal{B}$ is reasonable, then the pair $(!(\overline{\sigma}),\ ?(\sigma))$ is not in $\mathcal{B}$, and so σ and $\overline{\sigma}$ are not in $\mathcal{B}$-mutual compliance.

Since $\mathsf{unfold}(\sigma)$ performs inputs, while $\mathsf{unfold}(\overline{\sigma})$ performs outputs, and $\mathcal{B}$ is a reasonable relation, condition ii) above ensures that $(\,\mathsf{unfold}(\overline{\sigma}), \mathsf{unfold}(\sigma)\,) \notin \mathcal{B}$, so condition i) implies that $(\overline{\sigma},\ \sigma) \notin \mathcal{B}$. This implies that $!(\overline{\sigma}) \not\bowtie_{\mathcal{B}} ?(\sigma)$, and so $\sigma \parallel \overline{\sigma} \xrightarrow{\tau}_{\mathcal{B}} \mathsf{unfold}(\sigma) \parallel \mathsf{unfold}(\overline{\sigma}) \stackrel{\tau}{\not\longrightarrow}_{\mathcal{B}}$. But this means that $\sigma \not\dashv^{\mathcal{B}}_{\text{p2p}} \overline{\sigma}$ because neither $\mathsf{unfold}(\sigma)$ nor $\mathsf{unfold}(\overline{\sigma})$ perform $\checkmark$. ◻

In view of the previous example, we introduce a function to syntactically manipulate session contracts, whereby the result of manipulating ρ is a session contract in mutual compliance with ρ, at least for preorders $\mathcal{B}$s. In view of the encoding $\mathcal{M}$, this syntactic transformation applies equally well to session types.

Definition 6 ([Complement] [6]).
Let $\mathsf{cplmt} : L_{\mathsf{SCts}} \longrightarrow L_{\mathsf{SCts}}$ *be defined inductively as follows,*

$$\begin{aligned}
&\mathsf{cplmt}(1) = 1, \quad \mathsf{cplmt}(x) = x, && \mathsf{cplmt}(\mu x.\sigma) = \mu x.\mathsf{cplmt}(\sigma\lfloor^{\mu x.\sigma}/_{x}\rfloor),\\
&\mathsf{cplmt}(!(\sigma'').\sigma') = ?(\sigma'').\mathsf{cplmt}(\sigma'), && \mathsf{cplmt}(?(\sigma'').\sigma') = !(\sigma'').\mathsf{cplmt}(\sigma'),\\
&\mathsf{cplmt}(\textstyle\sum_{i\in I} ?\mathtt{l_i}.\sigma_i) = \bigoplus_{i\in I} !\mathtt{l_i}.\mathsf{cplmt}(\sigma_i), && \mathsf{cplmt}(\textstyle\bigoplus_{i\in I} !\mathtt{l_i}.\sigma_i) = \sum_{i\in I} ?\mathtt{l_i}.\mathsf{cplmt}(\sigma_i)
\end{aligned}$$

We say that $\mathsf{cplmt}(\sigma)$ *is the* complement *of* σ. ◻

In this definition the application of $\lfloor^{\sigma}/_{x}\rfloor$ to σ' stands for the substitution of σ in place of x in the message fields that appear in σ'; this is called *inner substitution* in [6]. The formal definition for our contracts is in [5, Appendix A].

Example 5. Suppose that $\sigma = \mu x.?(x).x$, then $\mathsf{cplmt}(\sigma) = \mu x.!(\sigma).x$. Observe that, intuitively, the input of σ depends on σ itself. The application of cplmt results in a contract which does not show that dependency, in that the output of $\mathsf{cplmt}(\sigma)$ does not depend on $\mathsf{cplmt}(\sigma)$.

Let us check a more involved example. We show how cplmt acts on session contracts. Let $\sigma = \mu x.\mu y.!(y)!(x).y$, and $\sigma' = \mu y.!(y)!(\sigma).y$. By definition,

$$\begin{aligned}\mathsf{cplmt}(\sigma) &= \mu x.\mathsf{cplmt}((\mu y.!(y).!(x).y)\lfloor^{\sigma}/_{x}\rfloor)\\ &= \mu x.\mathsf{cplmt}(\sigma')\\ &= \mu x.\mu y.\mathsf{cplmt}((!(y).!(\sigma).y)\lfloor^{\sigma'}/_{y}\rfloor)\\ &= \mu x.\mu y.\mathsf{cplmt}(!(\sigma').!(\sigma).y)\\ &= \mu x.\mu y.?(\sigma').?(\sigma).y\end{aligned}$$

Here again note that the contacts used in the input fields of $\mathsf{cplmt}(\sigma)$ are not defined in terms of $\mathsf{cplmt}(\sigma)$. □

In the previous example the contract σ and its complement are syntactically quite different objects, the complement being syntactically more complicated than σ,

$$\mathsf{cplmt}(\sigma) = \mu x.\mu y.?(\mu y.!(y)!(\sigma).y).?(\mu x.\mu y.!(y)!(x).y).y$$

What matters, though, are the behaviours of σ and of its complement. Those two behaviours are in $\mathcal{B}$-mutual compliance for every preorder $\mathcal{B}$. What was just argued for σ and its complement is true for every contract; the proof of it uses the commutativity of unfold and cplmt.

Proposition 2. *[Unfolding and complement commute]*
For every contract σ, $\mathsf{cplmt}(\mathsf{unfold}(\sigma)) = \mathsf{unfold}(\mathsf{cplmt}(\sigma))$.

Theorem 4. *For every preorder on contracts $\mathcal{B}$, $\rho \dashv^{\mathcal{B}}_{\text{P2P}} \mathsf{cplmt}(\rho)$ for every session contract ρ.*

In the full version of the paper [5] we argue that the notion of *complement* of a session type, Definition 6, in addition to being indispensable in the proof of Theorem 4, can also have a significant impact on type-checking systems for session types. For example the program P in Example 2 from the Introduction cannot be typed using the type-checking rules from [23]; the difficulty is the use of the duality operator $\overline{T}$ in the rule [CRes] on page 14. The bulk of the argument is that the dual of $\overline{\mu X.\,![\,X\,];\text{END}}$, that is $\mu X.\,?[\,X\,];\text{END}$, is not equivalent to $?[\,\mu X.\,![\,X\,];\text{END}\,];\text{END}$, and this hinders the necessary application of [CRes]. However we exhibit a type inference if instead $\mathsf{cplmt}(T)$ were used: the complement of $\mu X.\,![\,X\,];\text{END}$, namely $\mu X.\,?[\,\mu X.\,![\,X\,];\text{END}\,];\text{END}$, is equivalent to $?[\,\mu X.\,![\,X\,];\text{END}\,];\text{END}$, and this allows us to apply rule [CRes].

5 Related work

In this paper we proposed a new behavioural model for recursive higher-order session types [15], which is fully-abstract with respect to the subtyping relation [13]. The denotation of a type consists of the set of higher-order contracts with which it complies, when it in turn is viewed as a contract. We use a novel notion of compliance, called *peer compliance*, which is also parametrised with a particular decidable relation $\mathcal{B}_0$, used for comparing higher-order contracts which are supplied by one peer in order to satisfy the higher-order contract required by it's partner. Moreover this relation $\mathcal{B}_0$ is the maximal solution to a natural behavioural equation over contracts.

Contracts for web-service: First-order contracts for web-services and an operationally defined contract compliance have been proposed first in [16], where the compliance is defined in terms of the LTS of contracts, and then, in the style of testing theory [10], the sub-contract preorder is defined using the compliance. All the subsequent works - including this paper - adhere to that style.

The most recent accounts of first-order contracts for web-services are [19,9]. A striking difference between the two papers is the treatment of infinite behaviours. In [19] infinite behaviours are expressed by recursive contracts, whereas in [9] there is no recursive construct, $\mu X.-$, and the theory accounts for infinite behaviours by using a *coinductively* defined language. Our treatment of infinite behaviours follows the lines of [19].

Session types: Recursive higher-order session types appeared first in [15], where also the definition of type duality that we reported in Section 4 has been proposed. The authors of [15] argue in favour of program abstractions, that help programmers structure the interaction of processes around sessions. The proposed result is that a "typable program never reduces into an error" (see Theorem 5.4 (3) of [15]). In [23, pag. 86, paragraph 4], though, it is shown that that result is not true, that is the type system of [15] does not satisfy type-safety. The authors of [23] amend the type system of [15], thereby achieving type-safety (see Theorem 3.4 of [23]).

Subtyping for recursive higher-order session types has been introduced in [13], along with a *coinductive* definition of the duality. In addition to the standard type-safety result (Theorem 2), the authors show also a type-checking algorithm which they prove sound (Theorem 5) wrt the type system. The proof of completeness, though, relies on a relation between the inductive and the coinductive dualities (Proposition 5 there) which in general is false; a counter example is provided by the session type $\mu X.\,![\,X\,];\textsc{end}$. The consequence is that there is the possibility that the algorithm of [13], if employed in more general settings, may reject programs which are well-typed.

An alternative "fair" subtyping has been proposed recently in [20]. There session types are higher-order and recursive, their operational semantics is defined by parametrising the interactions of session types on pre-subtyping relations, and the fair subtyping is defined as a greatest fixed point (Definition 2.4). In our development we adopted the same technique as [20]. However, our aim was to model the standard subtyping of [13], while Padovani focuses on the properties of his new fair subtyping.

Models of Gay & Hole subtyping: The first attempt to model the Gay & Hole subtyping of [13] in terms of a compliance preorder appeared in [17]. For a comparison of that research and our work the reader is referred to [4]. The authors of [1] have shown

the first sound model of this subtyping restricted to first-order session types, by using a subset of contracts for web-services, a mutual compliance, called *orthogonality*, and the preorder generated by it. The $\mathcal{B}$-peer compliances we used in this work generalises to parametrised LTS the orthogonality of [1].

Following the approach of [1], in [4] we have shown a fully-abstract model of the subtyping for first-order session types, but using the standard asymmetric compliance and an intersection of the obvious server and client preorders. An alternative definition of the model proposed in [4] can be found in [3, Chapter 5], where the must testing of [10] is used in place of the compliance.

Semantic subtyping: We view our main result as a behavioural or *semantic* interpretation of Gay & Hole subtyping. There is an alternative well-developed approach to semantic theories of types and subtyping [12] in which the denotation of a type is given by the set of values which inhabit it, and subtyping is simply subset inclusion. This apparent simplicity is tempered by the fact that for non-trivial languages, such as the pi-calculus [7], there is a circularity in the constructions due to the fact that determining which terms are values depends in turn on the set of types. This circularity is broken using a technique called *bootstrapping* or *stratification*, essentially an inductive approach. The research using this approach which is closest to our results on Gay & Hole subtyping may be found in [8]; this contains a treatment of a very general language of session types, an extension of Gay & Hole types. But there are essential differences. The most important is that their model does not yield a semantic theory of Gay & Hole subtyping. Their subtyping relation, $\leq$, is defined via an LTS generated by considering the transmission of values rather than session types; effectively subtyping is not allowed on messages. The resulting subtyping is very different than our focus of concern, the Gay & Hole subtyping relation $\leqslant$. For example the preorder $\leq$ has bottom elements, in contrast to $\leqslant$, and $?[\,\texttt{Int}\,];\text{END} \leqslant ?[\,\texttt{Real}\,];\text{END}$ whereas $?[\,\texttt{Int}\,];\text{END} \not\leq ?[\,\texttt{Real}\,];\text{END}$. The particular use of *stratification* (Theorem 2.6) is also complex, and rules out the use of session types such as $\mu X.\,![\,X\,];\text{END}$. Finally they use as types infinite regular trees whereas we prefer to work directly with recursive terms, as proposed in [13]; for example this allows us to discuss the inadequacies of the type-checking rules of [23].

Nevertheless the extended language of sessions types of [8] is of considerable significance. It would be interesting to see if it can be interpreted behaviourally using our co-inductive approach, particularly endowed with a larger subtyping preorder more akin to the standard Gay & Hole relation [13].

Acknowledgements The authors would like to thank the reviewers, and reviewers of a previous version of this paper, for their insightful comments and questions.

References

1. Barbanera, F., de'Liguoro, U.: Two notions of sub-behaviour for session-based client/server systems. In: Kutsia, T., Schreiner, W., Fernández, M. (eds.) PPDP. pp. 155–164. ACM (2010)
2. Barbanera, F., de' Liguoro, U.: Sub-behaviour relations for session-based client/server systems (2013), submitted for publication
3. Bernardi, G.: Behavioural Equivalences for Web Services. Ph.D. thesis, Trinity College Dublin (2013), available at https://software.imdea.org/~giovanni.bernardi

4. Bernardi, G., Hennessy, M.: Modelling session types using contracts. In: Ossowski, S., Lecca, P. (eds.) SAC. pp. 1941–1946. ACM (2012)
5. Bernardi, G., Hennessy, M.: Using higher-order contracts to model session types. CoRR abs/1310.6176 (2013)
6. Bono, V., Padovani, L.: Typing copyless message passing. Logical Methods in Computer Science 8(1) (2012)
7. Castagna, G., De Nicola, R., Varacca, D.: Semantic subtyping for the pi-calculus. Theor. Comput. Sci. 398(1-3), 217–242 (2008)
8. Castagna, G., Dezani-Ciancaglini, M., Giachino, E., Padovani, L.: Foundations of session types. In: Porto, A., López-Fraguas, F.J. (eds.) PPDP. pp. 219–230. ACM (2009)
9. Castagna, G., Gesbert, N., Padovani, L.: A theory of contracts for web services. ACM Trans. Program. Lang. Syst. 31(5), 1–61 (2009), supersedes the article in POPL '08
10. De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science 34, 83–133 (1984)
11. Dezani-Ciancaglini, M., de'Liguoro, U.: Sessions and session types: An overview. In: Laneve, C., Su, J. (eds.) WS-FM. Lecture Notes in Computer Science, vol. 6194, pp. 1–28. Springer (2009)
12. Frisch, A., Castagna, G., Benzaken, V.: Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. J. ACM 55(4), 19:1–19:64 (Sep 2008), http://doi.acm.org/10.1145/1391289.1391293
13. Gay, S.J., Hole, M.: Subtyping for session types in the pi calculus. Acta Inf. 42(2-3), 191–225 (2005)
14. Hoare, C.A.R.: Communicating sequential processes. Prentice-Hall (1985)
15. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP. Lecture Notes in Computer Science, vol. 1381, pp. 122–138. Springer (1998)
16. Laneve, C., Padovani, L.: The must preorder revisited. In: Proceedings of the 18th international conference on Concurrency Theory. pp. 212–225. Springer-Verlag, Berlin, Heidelberg (2007), http://portal.acm.org/citation.cfm?id=1421822.1421826
17. Laneve, C., Padovani, L.: The pairing of contracts and session types. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. Lecture Notes in Computer Science, vol. 5065, pp. 681–700. Springer (2008)
18. Milner, R.: Communication and concurrency. PHI Series in computer science, Prentice Hall (1989)
19. Padovani, L.: Contract-based discovery of web services modulo simple orchestrators. Theor. Comput. Sci. 411(37), 3328–3347 (2010)
20. Padovani, L.: Fair Subtyping for Multi-Party Session Types. In: Proceedings of the 13th Conference on Coordination Models and Languages. vol. LNCS 6721, pp. 127–141. Springer (2011)
21. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Maritsas, D.G., Philokyprou, G., Theodoridis, S. (eds.) PARLE. Lecture Notes in Computer Science, vol. 817, pp. 398–413. Springer (1994)
22. Vasconcelos, V.T.: Fundamentals of session types. Inf. Comput. 217, 52–70 (2012)
23. Yoshida, N., Vasconcelos, V.T.: Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. Electr. Notes Theor. Comput. Sci. 171(4), 73–93 (2007)