USING HIGHER-ORDER CONTRACTS TO MODEL SESSION TYPES

GIOVANNI BERNARDI [a] AND MATTHEW HENNESSY [b]

[a] IMDEA Software Institute, Madrid, Spain
  e-mail address: bernargi@tcd.ie

[b] School of Statistics and Computer Science, O'Reilly Institute, Trinity College, Dublin 2, Ireland
  e-mail address: matthew.hennessy@scss.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 meaning of a contract is given in terms of the set of contracts with which it complies. A crucial step in the proof of full-abstraction is showing that every contract has at least one other contract with which it complies.

1. INTRODUCTION

The purpose of this paper is to show that session types [THK94, HVK98, DCd09] can be given a fully-abstract behavioural interpretation using web-service contracts [Pad10, CGP09]. 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. Moreover, to prove the completeness of the model we introduce a novel form of syntactic duality for higher-order contracts, which we call *peer-duality*. We believe that this *peer-duality*, when applied to session types, captures the intuition of *complementary behaviour* more faithfully than the standard notion of type duality from [HVK98].

The current paper is the full-version of the extended abstract [BH14]. It contains proofs for the various results, more explanations, and more examples. It also corrects a mistake in [BH14], which occurred when defining our novel form of syntactic duality for higher-order contracts. This will be explained in detail in Section 5.

Session types: The interactions between processes in a complex distributed system often follow a pre-ordained pattern. Session types [THK94, HVK98] have been proposed as a mechanism for concisely describing and structuring these interactions; they have been extensively studied [DCd09] and are being put into practice [HMM$^+$12]. As a simple example consider a system consisting of two entities

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

which first exchange a new private communication channel or *session*, s, over the public address of the store $\texttt{urls}$; using the conventions of [GH05] 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, each using its own exclusive endpoint of the private session s.

One form that the session type S could take is

$$\begin{aligned}
\texttt{?(Id)}.\,\&\langle\,\texttt{l}_1 \colon\ &\texttt{?(Addr).!(Int)}.T, \\
\texttt{l}_2 \colon\ &\texttt{?(Addr).!(Int)}.T, \\
\texttt{l}_3 \colon\ &\texttt{?(Addr).!(Int)}.T\,\rangle
\end{aligned} \tag{1.1}$$

where $\texttt{Int}, \texttt{Addr}, \texttt{Id}$ are some base types of integers, addresses and credentials respectively, and $\&\langle\,\texttt{l}_1 \colon S_1, \ldots, \texttt{l}_n \colon S_n\,\rangle$ is a *branch* type which accepts a choice between interaction on any of the predefined labels $\texttt{l}_i$, followed by the interaction described by the residual type S_i. Thus (1.1) 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 three commodities labelled by $\texttt{l}_1, \texttt{l}_2, \texttt{l}_3$, (iii) followed by the receipt of an address, and (iv) the transmission of a price, of type $\texttt{Int}$; 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 (1.1) above is

$$\begin{aligned}
\texttt{!(Id)}.\,\oplus\langle\,\texttt{l}_1 \colon\ &\texttt{!(Addr).?(Int)}.T', \\
\texttt{l}_2 \colon\ &\texttt{!(Addr).?(Int)}.T', \\
\texttt{l}_3 \colon\ &\texttt{!(Addr).?(Int)}.T'\,\rangle
\end{aligned} \tag{1.2}$$

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 $\oplus\langle\,\texttt{l}_1 \colon S_1, \ldots, \texttt{l}_k \colon S_k\,\rangle$, which allows the process executing the role described by the type to choose one among the labels $\texttt{l}_i$. These two principles lead to a general definition of the *dual* of a session type T, denoted $\overline{T}$ in [THK94, HVK98].

In order to allow flexibility to the processes fulfilling the roles described by these types a subtyping relation between session types, $T \leqslant S$, is essential; see [GH05] for a description of the crucial role played by subtyping. Intuitively $T \leqslant S$ 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. However, to the best of our knowledge, subtyping for session types has only ever been given a purely syntactic definition; in Definition 2.3 of Section 2 we slightly generalise the standard definition of [GH05], so as to account also for base types such as $\texttt{Int}$ and $\texttt{Bool}$.

Recursive types: Sessions over which interactions have reached completion are described by the type END. Some sessions, though, may allow interactions between their endpoints to go on indefinitely. This is required for instance by processes that offer services or methods. To accommodate this, recursion has been added in some process calculi, for instance in [HVK98, YV07, DH11]. To show instances of such processes below, and also in Section 5.4, we use the syntax of [YV07]. The only

non-standard constructs in that syntax are the two which describe session delegation, that is `throw` and `catch`. The process `throw` $\kappa'[\kappa]; P$ writes the endpoint κ over the endpoint κ', and continues as P. The process `catch` $\kappa'(z)$ `in` $X[Q]$, on the other hand, inputs a name, say κ, over the endpoint κ', and continues as the process $Q[z \mapsto \kappa]$.

Example 1.1. [An ever-lasting session]
Here we use the syntax of [YV07] to describe processes.

$$
\begin{aligned}
D_s \;\; &= \;\; X(y) := y \rhd \{\, \texttt{plus}: y?(x) \text{ in } y?(z) \text{ in } y![x+z].X[y] \, [\!] \\
&\qquad\qquad\qquad\quad \texttt{pos}: y?(z) \text{ in } y![z>0].X[y] \, \}
\end{aligned}
$$

$$
\begin{aligned}
D_c \;\; &= \;\; Y(x) := x \lhd \{\, \texttt{pos}: x![\,random()\,].x?(z) \text{ in } Y[x] \, \}. \\
P \;\; &= \;\; (\nu\kappa)(\texttt{def } D_s \text{ in def } D_c \text{ in } X[\kappa^+] \parallel Y[\kappa^-])
\end{aligned}
$$

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 [GH05].

The type T that describes the behaviour of D_s on an endpoint k^+ is intuitively a solution of the equation

$$
T = \&\langle \texttt{plus}: \texttt{?(Int).?(Int).!(Int)}.T, \; \texttt{pos}: \texttt{?(Int).!(Int)}.T \rangle
$$

A natural way to express solutions to such equations is to use recursive types. For instance we could take the type T required above to be

$$
\mu X. \&\langle \texttt{plus}: \texttt{?(Int).?(Int).!(Int)}.X, \; \texttt{pos}: \texttt{?(Int).!(Int)}.X \rangle
$$

The type equations that require recursive terms arise naturally in presence of recursive processes. The presence of recursive types justifies the coinductive definition of the subtyping (see Definition 2.3), which follows the approach of [PS96].

Contracts: Web services [Pad10, CGP09] 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 [Mil89b, Hoa85], and indeed the theory of these process calculi have been used to give a behavioural interpretation of contracts and the sub-contract preorder, [Pad10, LP07].

Contracts are very similar, at least syntactically, to sessions types; for example (1.2) above can very easily be read as the following process description from CCS,

$$
!(\texttt{Id}).(?l_1.?\texttt{Addr}.!\texttt{Int}.\texttt{cnt}' + ?l_2.?\texttt{Addr}.!\texttt{Int}.\texttt{cnt}' + ?l_3.?\texttt{Addr}.!\texttt{Int}.\texttt{cnt}')
$$

In fact in Section 3 we give a straightforward translation $\mathcal{M}$ from the language of session contracts to that of contracts. Via the mapping $\mathcal{M}$ it should therefore be possible to give a behavioural interpretation to session types, thereby explaining how session types determine process behaviour, at least along individual sessions. Indeed steps in this direction have already been made in [Bd10, BH12] 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 costumer **cstmr** is replaced by **girlf** and there are now four components:

$$(\nu s)\,(\nu p)\,(\nu b)\,(\texttt{urls}![\,s^+\,].\texttt{urlb}![\,p^+\,].\texttt{urlb}![\,b^+\,].\textbf{girlf}\,\|$$
$$\texttt{urls}?(s^+ : S\,).\textbf{store}\,\|$$
$$\texttt{urls}?(p^+ : S_p).\textbf{bank}\,\|$$
$$\texttt{urlbf}?(b^+ : S_b).\textbf{boyf})$$

Here we model a scenario in which a girlfriend opens a connection to a store, lets her boyfriend choose a commodity, and then pays for the present. 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

$$?(T_1).!(T_2).\textsc{end} \tag{1.3}$$

where in turn T_1 is the session type $\oplus\langle\, \mathtt{l}_1 : ?(\texttt{Addr}).\textsc{end}\,\rangle$ 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 1.2. [Everlasting generation of finite sessions]
Consider the process P defined as follows,

$$D \;\; := \;\; X(x, y) = (\nu\kappa_\mathbf{f})\,(\texttt{throw}\;x[\,\kappa_\mathbf{f}^+\,];\mathbf{0}\,\|\,\texttt{catch}\;y(z)\,\texttt{in}\,X[\,z, \kappa_\mathbf{f}^-\,])$$

$$P \;\; = \;\; (\nu\kappa_\mathbf{o})\,(\texttt{def}\;D\;\texttt{in}\;X[\,\kappa_\mathbf{o}^+, \kappa_\mathbf{o}^-\,])$$

Intuitively, at each iteration the code $X[\,\kappa_\mathbf{o}^+, \kappa_\mathbf{o}^-\,]$ has the two endpoints of a pre-existing session, $\kappa_\mathbf{o}$, generates a new *fresh* session, $\kappa_\mathbf{f}$, 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 [YV07] 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 $\mu X.!(X).\textsc{end}$. Such types are forbidden in [BdL13] but they are allowed in the typing systems of [HVK98, GH05, YV07, Vas12]; we discuss the type inference in Section 5.4.

   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 of (1.3) above is mapped by $\mathcal{M}$ to the contract $?(!\mathtt{l}_1.?(\texttt{Addr}).1).?(\texttt{cnt}_2).1$, where $\texttt{cnt}_2 = \mathcal{M}(T_2)$. 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 $\sqsubseteq$ with the property that, for all session types,

$$S \preccurlyeq T \text{ if and only if } \mathcal{M}(S) \sqsubseteq \mathcal{M}(T) \tag{1.4}$$

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 this sub-contract preorder is defined; we will refer to it as the *peer sub-contract preorder*. Intuitively $\sigma_1 \sqsubseteq \sigma_2$, 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 $\rho \dashv\vdash_{\text{p2p}} \sigma$, 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 relation, which is symmetric, *mutual* or *peer* compliance, as both participants are required to attain a *happy* state simultaneously. This is in contrast to [CGP09, Pad10, BH12] 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

$$\rho \parallel \sigma \xrightarrow{\tau} \rho' \parallel \sigma' \qquad (1.5)$$

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 right-hand side of the parallel constructor $\parallel$ we have a contract guaranteed by a process which will accept a datum which can be used as a real; on the left-hand side there is a contract guaranteed by a process that supplies an $\texttt{Int}$. Since we are assuming that integers can be interpreted as reals, that is $\texttt{Int} \leqslant_b \texttt{Real}$, we know that an interaction described by the judgement above takes place.

However it is unclear when an interaction of the form

$$\texttt{!}(\sigma_1).\rho' \parallel \texttt{?}(\sigma_2).\sigma' \xrightarrow{\tau} \rho' \parallel \sigma' \qquad (1.6)$$

should take place. Here on the right is a contract satisfied by a process which provides a session endpoint that satisfies the contract σ_2; on the left is a contract satisfied by one which is willing to accept any session endpoint which guarantees the contract σ_1. Intuitively the interaction should be allowed if σ_1 is a sub-contract of σ_2, that is $\sigma_1 \sqsubseteq \sigma_2$. However the whole purpose of defining the judgements (1.5) above is in order to define the preorder $\sqsubseteq$; there is a circularity in our arguments.

We break this circularity by supposing a predefined sub-contract preorder $\mathcal{B}$ and allowing the interaction (1.6) 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 $\sigma \dashv\vdash_{\text{p2p}}^{\mathcal{B}} \rho$ 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, (1.4) above, by showing:

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

This particular preorder $\mathcal{B}_0$ which we construct, and which has been referred to in (1.4) above as $\sqsubseteq$, has a natural behavioural interpretation. It satisfies the behavioural equation

$$\sigma_1 \ \mathcal{B}_0 \ \sigma_2 \text{ if and only if } \sigma_1 \sqsubseteq^{\mathcal{B}_0} \sigma_2 \tag{1.7}$$

Moreover it is the largest preorder between higher-order contracts which satisfies (1.7).

The proof of (1.7) depends on an alternative syntactic characterisation of the set-based preorders $\sqsubseteq^{\mathcal{B}}$. The proof of this syntactic characterisation relies in turn on a crucial property of the peer-compliance relation:

for every contract σ there exists a complementary contract ρ which complies with it, $\sigma \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \rho.$ $(\star)$

However constructing such complementary contracts is not straightforward. One possibility is to use the notion of the *dual* of a session type, [THK94, HVK98], already discussed informally on page 2. A formal definition may be found in Figure 5, and is readily adapted to contracts via the translation $\mathcal{M}$; specifically we can define $\overline{\sigma}$ to be $\mathcal{M}(\overline{\mathcal{M}^{-1}(\sigma)})$.

However there are contracts σ which do not comply with their duals, $\sigma \not\dashv\vdash^{\mathcal{B}}_{\text{P2P}} \overline{\sigma}$. Moreover these are not esoteric contracts but occur naturally when modelling reasonable processes. A typical example is the contract $\mu x.?(x).1$, corresponding to the session type $\mu X.!(X).\text{END}$ needed to type the process in Example 1.2. In Example 5.2 we explain why this contract does not comply with its dual, assuming $\mathcal{B}$ satisfies some minimal requirements.

The problem occurs with recursive contracts in which recursion variables occur in message fields. Indeed in Theorem 5.7 we show that $\sigma \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \overline{\sigma}$ whenever σ has no such occurrences of recursion variables; again subject to minor conditions on $\mathcal{B}$.

However for arbitrary contracts we need a more general method of constructing a complementary contract, which for example applies to useful contracts such as $\mu x.?(x).1$. In [BH14] we used a function, $\text{cplmt}(-)$, to fulfil this role; the same function was independently proposed in [BP12], although with a different purpose in mind. But unfortunately this has the same defect as duality: there are also contracts σ such that $\sigma \not\dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{cplmt}(\sigma)$; see Example 5.21.

Here we propose another, more complicated, function $\text{prdual}(-)$, see Definition 5.16; we refer to $\text{prdual}(\sigma)$ as the *peer-dual* of the contract σ. The intuitive idea is that $\text{prdual}(\sigma)$

- first syntactically transforms σ into another contract σ_{ok} which has no offending recursion variables, but is in some sense still functionally equivalent to the original σ
- then returns the standard dual of σ_{ok}, namely $\overline{\sigma_{\text{ok}}}$.

In Theorem 5.17 we prove that $\sigma \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{prdual}(\sigma)$ for every session contract σ, thus establishing $(\star)$ above.

Paper structure: The remainder of the paper is organised as follows. In Section 2 we recall the standard theory of session types, while Section 3 is devoted to our exposition of higher-order contracts and our novel notion of parametrised peer sub-contract preorder $\sqsubseteq^{\mathcal{B}}$. In Section 3.2 we show that, 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. Using this syntactic characterisation we can develop enough properties of the preorders $\sqsubseteq^{\mathcal{B}}$ to ensure the existence of the particular preorder $\mathcal{B}_0$ alluded to in (1.7) above; this is the topic of Section 4.

As we have already stated, this result depends on being able to construct the complement of a contract. Our proposal, $\text{prdual}(\sigma)$, is discussed separately in Section 5. This section also contains explanations of the deficiencies of the previous proposals $\overline{\sigma}$, and $\text{cplmt}(\sigma)$. It then finishes, in Section 5.4, with a discussion of how our proposed operator $\text{prdual}(-)$ can be used to improve on the type-checking systems for session types in papers such as [YV07, Vas12].

T, S	::=		**Session types**
		END	*Terminated session*
		$?(M).T$	*Input*
		$!(M).T$	*Output*
		$\&\langle 1_1 : T_1, \ldots, 1_n : T_n \rangle$	*Branch*, $n \geq 1$
		$\oplus\langle 1_1 : T_1, \ldots, 1_n : T_n \rangle$	*Choice*, $n \geq 1$
		$\mu X.T$	*Recursive session type*
		X	*Type variable*
M, N	::=		**Message types**
		T	*Session Type*
		t	*Base type*
t	::=		**Base types**
		$\mathtt{Id}, \mathtt{Addr}, \mathtt{Int}, \mathtt{Real}, \ldots$	

Figure 1: Grammar of types

Related work is then discussed in Section 6, and the appendices contain some standard material and some minor technical results.

2. Session types

Here we recall, using the notation from [GH05], the standard theory of subtyping for session types. The grammar for the language L_{STyp} of session type terms is given in Figure 1. That grammar uses a collection of unspecified base types BT, of which we enumerate a sample. It also uses a denumerable set of labels, $\mathtt{L} = \{1_1, 1_2, 1_3, \ldots\}$, in the *branch* and *choice* constructs. Recall from the Introduction that $\&\langle 1_1 : S_1, \ldots, 1_n : S_n \rangle$ offers different possible behaviours based on a set of labels $\{1_1, 1_2, 1_3, \ldots 1_n\}$ while $\oplus\langle 1_1 : S_1, \ldots, 1_n : S_n \rangle$ 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*, in other words the terms that contain only variables which are not free, and that appear at least after one non-recursive type constructor. The definitions of both concepts are standard, and they may be found in Appendix A.

Subtyping is defined coinductively and uses some unspecified subtyping preorder $\leqslant_b$ between base types, a typical example being $\mathtt{Int} \leqslant_b \mathtt{Real}$, meaning that an integer may be supplied where a real number is required. Intuitively subtyping between session types is determined by two principles:

(a) *Branch*: a branch type T_1 can be replaced with a branch type T_2 that allows *more* choices than T_1.
(b) *Choice*: a choice type T_1 can be replaced with a choice type T_2 that allows *fewer* choices than T_1.

We give two examples of these principles.

Example 2.1. Let BARTENDER = $\&\langle$ espresso: $T_1\rangle$ and

$$\text{FANCYBARTENDER} = \&\langle \text{espresso}: T_1',$$
$$\text{deka}: T_2',$$
$$\text{double} - \text{espresso}: T_3'\rangle$$

The type BARTENDER accepts the choice of only one option, espresso, so all the customers satisfied by BARTENDER, are satisfied by any other type that offers *at least* the label espresso. It follows that FANCYBARTENDER satisfy all the customers satisfied by the BARTENDER. This is formalised by the subtyping, which relates the two types

$$\text{BARTENDER} \leqslant \text{FANCYBARTENDER}$$

as long as also the continuations T_1 and T_1' are related (ie. $T_1 \leqslant T_1'$).

Example 2.2. Let ITALIANCUSTOMER describe the different coffees that a process may want to order when interacting with a bar tender.

$$\text{ITALIANCUSTOMER} = \oplus\langle \text{espresso}: T_1,$$
$$\text{deka}: T_2,$$
$$\text{double} - \text{espresso}: T_3\rangle$$

All the bar tenders that are able to accept this range of choices, have to offer *at least* the three labels that appear in ITALIANCUSTOMER. Now consider the type

$$\text{CUSTOMER} = \oplus\langle \text{espresso}: T_1'\rangle$$

Since CUSTOMER chooses among fewer options than ITALIANCUSTOMER, a process that behaves according to CUSTOMER can be used in place of one that behaves as prescribed by ITALIANCUSTOMER. This is formalised by the subtyping relation as follows,

$$\text{ITALIANCUSTOMER} \leqslant \text{CUSTOMER}$$

if also the continuations are related (ie. $T_1 \leqslant T_1'$).

Added to the principles sketched above are the standard covariant and contravariant requirements on the input and output constructs. Recursive types are handled by a standard function 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 $T[X \mapsto S]$, the syntactic substitution of the term S for all free occurrences of X in T. The details may be found in Appendix A.

Definition 2.3. [Subtyping]
Let $\mathcal{F}^{\leqslant} : \mathcal{P}(\text{STyp}^2) \longrightarrow \mathcal{P}(\text{STyp}^2)$ be the functional defined so that $(T, S) \in \mathcal{F}^{\leqslant}(\mathcal{R})$ whenever:
   (i) if unfold(T) = END then unfold(S) = END
  (ii) if unfold(T) = ?(t_1).S_1 then unfold(S) = ?(t_2).S_2 and $S_1 \mathcal{R} S_2$ and $\text{t}_1 \leqslant_\text{b} \text{t}_2$
 (iii) if unfold(T) = !(t_1).S_1 then unfold(S) = !(t_2).S_2 and $S_1 \mathcal{R} S_2$ and $\text{t}_2 \leqslant_\text{b} \text{t}_1$
 (iv) if unfold(T) = !(T_1).S_1 then unfold(S) = !(T_2).S_2 and $S_1 \mathcal{R} S_2$ and $T_2 \mathcal{R} T_1$
  (v) if unfold(T) = ?(T_1).S_1 then unfold(S) = ?(T_2).S_2 and $S_1 \mathcal{R} S_2$ and $T_1 \mathcal{R} T_2$
 (vi) if unfold(T) = $\&\langle \text{l}_1: T_1, \ldots, l_m: T_m\rangle$ then unfold(S) = $\&\langle \text{l}_1: S_1, \ldots, \text{l}_n: S_n\rangle$ where $m \leq n$
       and $T_i \mathcal{R} S_i$ for all $i \in [1, \ldots, m]$
 (vii) if unfold(T) = $\oplus\langle \text{l}_1: T_1, \ldots, l_m: T_m\rangle$ then unfold(S) = $\oplus\langle \text{l}_1: S_1, \ldots, \text{l}_n: S_n\rangle$ where $n \leq m$
       and $T_i \mathcal{R} S_i$ for all $i \in [1, \ldots, n]$

If $\mathcal{R} \subseteq \mathcal{F}^{\preccurlyeq}(\mathcal{R})$, then we say that R is a *type simulation*. The monotonicity of $\mathcal{F}^{\preccurlyeq}$ and the Knaster-Tarski Theorem [Tar55] ensure that there exists the greatest solution of the equation $\mathcal{R} = \mathcal{F}^{\preccurlyeq}(\mathcal{R})$; we call this solution the *subtyping*, and we denote it $\preccurlyeq$.

The cases ((ii)) and ((iii)) are not present in the original definition of [GH05], so our subtyping relation is in fact slightly more general than the one of that paper. Note the use of covariance in the first argument in ((ii)), ((v)) and the contravariance in ((iii)) and ((iv)); the intuitions of (a) and (b) above are reflected in ((vi)) and ((vii)) respectively.

In the next example we show how to prove that two types are related by the subtyping.

Example 2.4. Let $S = \mu X.?(X).X$ and $T = \mu Y.?(Y).?(Y).Y$. In this example we prove that

$$\mu X.?(X).X \preccurlyeq \mu Y.?(Y).?(Y).Y$$

Thanks to the Knaster-Tarski Theorem, we need only exhibit a prefixed point of the functional $\mathcal{F}^{\preccurlyeq}$, that contains the pair (S, T). Consider the following relation

$$\mathcal{R} = \{(S, T), (?(S).S, ?(T).?(T).T), (?(S).S, ?(T).T), (S, ?(T).T)\}$$

and let $A = \mathcal{F}^{\preccurlyeq}(\mathcal{R})$. The set inclusion $\mathcal{R} \subseteq A$ follows from case ((v)) of Definition (2.3).

Now that we have defined the subtyping relation $\preccurlyeq$, and shown a proof method for it, we discuss its meaning. In the context of π-calculus equipped with sessions, the relation $\preccurlyeq$ formalises two notions of safe substitutivity. Suppose that $\texttt{Int} \preccurlyeq_{\mathsf{b}} \texttt{Real}$. If $S \preccurlyeq T$, then both of the following are true:

(1) a session end-point κ_2 at type S may safely be used in place of a session endpoint κ_1 at type T. For example - using the standard π syntax - a process

$$P = c?(x \colon ?(\texttt{Real}).\textsc{end}).x?(y \colon \texttt{Real}).b![\, y/2 \,].\mathbf{0}$$

may safely receive along the channel c an endpoint κ_2 that has type $?(\texttt{Int}).\textsc{end}$, instead of the declared type $?(\texttt{Real}).\textsc{end}$. Intuitively, this is the case because $?(\texttt{Int}).\textsc{end} \preccurlyeq ?(\texttt{Real}).\textsc{end}$, and thus if P can read over x a value of type $\texttt{Real}$, then it can read also a value of type $\texttt{Int}$.

(2) a process P that uses an endpoint κ at type T may be safely used in place of a process Q that uses κ at type S. To sketch this phenomenon, we follow [GH05, Section 2] and discuss the following processes,

$$
\begin{aligned}
\mathsf{clientbody}(x) \quad &= \quad x \triangleleft \{\, \texttt{plus} \colon x![\, 2 \,].x![\, 3 \,].x?(u \colon \texttt{Int}).\mathbf{0} \,\}. \\
\mathsf{serverbody}_1(x) \quad &= \quad x \triangleright \{\, \texttt{plus} \colon x?(z \colon \texttt{Int}).x?(y \colon \texttt{Int}).x![\, y + z \,].\mathbf{0} \,\} \\
\mathsf{serverbody}_2(x) \quad &= \quad x \triangleright \{\, \texttt{plus} \colon x?(z \colon \texttt{Int}).x?(y \colon \texttt{Int}).x![\, y + z \,].\mathbf{0} \\
&\qquad\qquad\quad \texttt{mult} \colon x?(z \colon \texttt{Int}).x?(y \colon \texttt{Int}).x![\, y * z \,].\mathbf{0} \,\}
\end{aligned}
$$

The session type at which $\mathsf{serverbody}_1(x)$ employs x is $S = \&\langle\, \texttt{plus} \colon S' \,\rangle$, where

$$S' = ?(\texttt{Int}).?(\texttt{Int}).!(\texttt{Int}).\textsc{end},$$

while the type at which $\mathsf{serverbody}_2(x)$ uses x is $T = \&\langle\, \texttt{plus} \colon S', \texttt{mult} \colon S' \,\rangle$, and it is routine work to check that $S \preccurlyeq T$. Now observe that - at least intuitively - the process

$$Q_1 = (\nu\kappa)\,(\mathsf{clientbody}(\kappa^-) \parallel \mathsf{serverbody}_1(\kappa^+))$$

is well-typed. Thanks to $S \preccurlyeq T$ and subsumption, one can adapt the type derivation for Q_1 to prove that also the following process is well-typed,

$$Q_2 = (\nu\kappa)\,(\mathsf{clientbody}(\kappa^-) \parallel \mathsf{serverbody}_2(\kappa^+)).$$

ρ, σ ::=		**Higher-order session contracts**
	1	*Satisfied contract*
	$?\mathsf{t}.\sigma$	*Input*
	$!\mathsf{t}.\sigma$	*Output*
	$!(\sigma).\sigma$	*Contract output,*
	$?(\sigma).\sigma$	*Contract input,*
	$\sum_{i \in I} ?\mathsf{l}_i.\sigma_i$	*External sum I* non-empty, finite
	$\bigoplus_{i \in I} !\mathsf{l}_i.\sigma_i$	*Internal sum I* non-empty, finite
	$\mu x.\sigma$	*Recursive session contract*
	x	*Session contract variable*

Figure 2: Grammar of higher-order session contracts

Essentially, by knowing that Q_1 is well-typed and that $S \preccurlyeq T$, one can show that the process $\mathsf{serverbody}_1(x)$ can be safely replaced by the process $\mathsf{serverbody}_2(x)$, in the sense that also Q_2 is well-typed.

The argument in (2) above let us argue that $\mathsf{serverbody}_2(x)$ may be used where $\mathsf{serverbody}_1(x)$ is required. Since the behaviours of these processes are described respectively by the types S and T, we can reason directly on types, and argue that $S \preccurlyeq T$ means that processes adhering to the role dictated by T may be used where processes following the role dictated by S are required. Out 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* sub-contract preorder.

3. HIGHER-ORDER CONTRACTS

In the first section we define higher-order session contracts and explain the set-based subcontract preorder on them; this uses the notion of *peer compliance* between them. In the following section we show that this set-based preorder can be characterised by comparing the purely syntactic structure of contracts, up-to a parameter $\mathcal{B}$.

3.1. **Contracts and compliance.** The grammar for the language of contract terms L_{SCts} is given in Figure 1; there we assume the labels l_is to be pairwise distinct. 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.

The operational meaning of contracts is given by viewing them as processes from a simple process calculus and interpreting them as states in a (higher-order) labelled transition system. To this end let

$$\mathsf{Act} = \{\, ?\mathsf{l}, !\mathsf{l} \mid \mathsf{l} \in \mathsf{L} \,\} \cup \{\, ?\mathsf{t}, !\mathsf{t} \mid t \in \mathsf{BT} \,\} \cup \{\, ?(\sigma), !(\sigma), \mid \sigma \in \mathsf{SCts} \,\}$$

be the set of prefixes, ranged over by λ. We use $\mathsf{Act}_{\tau,\checkmark}$ to denote the set $\mathsf{Act} \cup \{\tau, \checkmark\}$ to emphasise that the special symbols τ and $\checkmark$ are not in Act. In Figure 3 we give a set of axioms which define judgements of the form

$$\sigma_1 \xrightarrow{\mu} \sigma_2$$

$$\frac{}{1 \xrightarrow{\checkmark}} \text{[A-Ok]} \qquad\qquad \frac{}{\lambda.\sigma \xrightarrow{\lambda} \sigma} \lambda \in \mathsf{Act} \ \text{[A-Pre]}$$

$$\frac{}{\bigoplus_{i\in I} !\mathtt{l_i}.\sigma_i \xrightarrow{\tau} !\mathtt{l_i}.\sigma_i} |I| > 1 \ \text{[A-Int]} \qquad\qquad \frac{}{\sum_{i\in I} ?\mathtt{l_i}.\sigma_i \xrightarrow{?\mathtt{l_i}} \sigma_i} \text{[A-Ext]}$$

$$\frac{}{\mu x.\sigma \xrightarrow{\tau} \sigma[x \mapsto \mu x.\sigma]} \text{[A-Unfold]}$$

Figure 3: The operational semantics of session contracts

where $\mu \in \mathsf{Act}_{\tau\checkmark}$ and $\sigma_1, \sigma_2 \in \mathsf{SCts}$. Note that that terms like $!\mathtt{l}.\sigma$ are singleton internal sums and we infer their semantics by using the rule for prefixes, [A-Pre]; for example $!\mathtt{l}.\sigma \xrightarrow{!\mathtt{l}} \sigma$. The rule [A-Unfold] uses a form of substitution of a closed term ρ for free occurrences of a variable x in a term σ, denoted $\sigma[x \mapsto \rho]$. In Appendix A we give a more general definition of the application of a substitution $\mathbf{s}$ to a term σ, denoted $\sigma\,\mathbf{s}$; then $\sigma[x \mapsto \rho]$ corresponds to $\sigma\mathbf{s}_x$, where $\mathbf{s}_x$ is a simple substitution which maps the variable x to the closed term ρ. We say that a contract σ is *stable* whenever $\sigma \xnot\xrightarrow{\tau}$.

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

$$\lambda_1 \bowtie_{\mathcal{B}} \lambda_2 = \begin{cases} \lambda_1 = !\mathtt{l}, \ \lambda_2 = ?\mathtt{l} \\ \lambda_1 = ?\mathtt{l}, \ \lambda_2 = !\mathtt{l} \\ \lambda_1 = !\mathtt{t_1}, \ \lambda_2 = ?\mathtt{t_2} & \mathtt{t_1} \leqslant_b \mathtt{t_2} \\ \lambda_1 = ?\mathtt{t_1}, \ \lambda_2 = !\mathtt{t_2} & \mathtt{t_2} \leqslant_b \mathtt{t_1} \\ \lambda_1 = !(\sigma_1), \ \lambda_2 = ?(\sigma_2) & \sigma_1 \ \mathcal{B} \ \sigma_2 \\ \lambda_1 = ?(\sigma_1), \ \lambda_2 = !(\sigma_2) & \sigma_2 \ \mathcal{B} \ \sigma_1 \end{cases}$$

Essentially the relation $\bowtie_{\mathcal{B}}$ treats $\mathcal{B}$ as a subtyping on contracts; note that by definition $\bowtie_{\mathcal{B}}$ is symmetric, for any $\mathcal{B}$.

The inference rules in Figure 4 are now straightforward; $\rho \parallel \sigma$ can proceed if either of the components σ, ρ can proceed independently, or if the components can interact, as dictated by $\bowtie_{\mathcal{B}}$.

We are now ready to define our version of *(peer) compliance*.

Definition 3.1. Let $C^{\mathrm{p2p}} : \mathcal{P}(\mathsf{SCts}^2) \times \mathcal{P}(\mathsf{SCts}^2) \longrightarrow \mathcal{P}(\mathsf{SCts}^2)$ be the rule functional defined so that $(\rho, \sigma) \in C^{\mathrm{p2p}}(\mathcal{R}, \mathcal{B})$ whenever both the following conditions hold:

$$\frac{\rho \xrightarrow{\tau} \rho'}{\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma} \text{ [I-Left]} \qquad \frac{\sigma \xrightarrow{\tau} \sigma'}{\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho \parallel \sigma'} \text{ [I-Right]}$$

$$\frac{\rho \xrightarrow{\lambda_1} \rho' \quad \sigma \xrightarrow{\lambda_2} \sigma'}{\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma'} \;\lambda_1 \bowtie_{\mathcal{B}} \lambda_2 \text{ [I-Synch]}$$

Figure 4: Interacting session contracts

(i) if $\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}}$ 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 $\mathcal{R} \subseteq C^{\text{p2p}}(\mathcal{R}, \mathcal{B})$, 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 $X = C^{\text{p2p}}(X, \mathcal{B})$; we call this solution the *$\mathcal{B}$-peer compliance*, and we denote it $\dashv\vdash^{\mathcal{B}}_{\text{p2p}}$.

The intuition here is that if $\rho \dashv\vdash^{\mathcal{B}}_{\text{p2p}} \sigma$ then pairs of processes satisfying these contracts can interact indefinitely, until such time that they can both simultaneously perform the success action $\checkmark$. So the interaction between them can continue indefinitely, even forever; but if further interaction is not possible then condition ((i)) ensures that both participants must have reached the *happy* state.

Example 3.2. We show the impact of the symmetric requirement in Definition 3.1((i)). Consider the contracts $\rho = \,!\text{Int}.1$, and $\sigma = \mu y.?\text{Int}.y$. While ρ requires just one interaction to reach the satisfied state 1, the contract σ supports an infinite number of interactions, but never reaches a satisfied state. Under the reasonable assumption that $\text{Int} \leqslant_b \text{Int}$, after one interaction the composition $\rho \parallel \sigma$ reduces to a stable state, in which the derivative of σ is not satisfied; therefore $\rho \not\dashv\vdash^{\mathcal{B}}_{\text{p2p}} \sigma$.

In general the properties of the compliance relation $\dashv\vdash^{\mathcal{B}}_{\text{p2p}}$ depends on those of the underlying relation $\mathcal{B}$, and of the first-order subtyping relation $\leqslant_b$, for example $!(0).1 \not\dashv\vdash^{\emptyset}_{\text{p2p}} ?(1).1$, while $!(0).1 \dashv\vdash^{\mathcal{B}}_{\text{p2p}} ?(1).1$ for $\mathcal{B} = \{(0, 1)\}$. The ability of a contract to comply with another depends not on its syntax but rather on its behaviour, which is determined by its operational semantics as given by the rules in Figure 3. To emphasise this point let us adapt the standard notion of *bisimulation equivalence* [Mil89a] to our setting.

Definition 3.3. [Strong bisimulation]
A relation $\mathcal{R} \subseteq \text{SCts} \times \text{SCts}$ is called a (strong) bisimulation if whenever $\sigma_1 \mathcal{R} \sigma_2$ then

(1) $\sigma_1 \xrightarrow{\checkmark}$ if and only if $\sigma_2 \xrightarrow{\checkmark}$, and
(2) for every $\mu \in \text{Act}_\tau$
   (a) $\sigma_1 \xrightarrow{\mu} \sigma_1'$ implies $\sigma_2 \xrightarrow{\mu} \sigma_2'$ for some σ_2', such that $\sigma_1' \mathcal{R} \sigma_2'$
   (b) conversely, $\sigma_2 \xrightarrow{\mu} \sigma_2'$ implies $\sigma_1 \xrightarrow{\mu} \sigma_1'$ for some σ_1', such that $\sigma_1' \mathcal{R} \sigma_2'$,
We write $\sigma_1 \sim \sigma_2$ whenever there is some bisimulation $\mathcal{R}$ such that $\sigma_1 \mathcal{R} \sigma_2$.

Our main interest in this strong form of bisimulation is encapsulated in Proposition 3.5, which in turn uses the next lemma.

Lemma 3.4. *Suppose $\rho_1 \sim \rho_2$. Then for every $\sigma \in \text{SCts}$ and every $\mathcal{B}$, $\rho_1 \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}}$ implies $\rho_2 \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}}$.*

Proof. The hypothesis $\rho_1 \parallel \sigma \xrightarrow{\tau}\!\!\!\!\!/\;_\mathcal{B}$ imply that both ρ_1 and σ are stable. Definition 3.3 ensures that ρ_2 is also stable. To prove that $\rho_2 \parallel \sigma \xrightarrow{\tau}\!\!\!\!\!/\;_\mathcal{B}$ it suffices to show that (a) if $\rho_2 \xrightarrow{\lambda}$ then $\sigma \xrightarrow{\lambda'}$ implies $\lambda \not\bowtie_\mathcal{B} \lambda'$. Suppose that $\rho_2 \xrightarrow{\lambda}$ for some λ, part ((a)) of Definition 3.3 and the hypothesis that $\rho_1 \sim \rho_2$ guarantee that $\rho_1 \xrightarrow{\lambda}$. The hypothesis that $\rho_1 \parallel \sigma \xrightarrow{\tau}\!\!\!\!\!/\;_\mathcal{B}$ and rule [I-Synch] in Figure 4 ensure that $\sigma \xrightarrow{\lambda'}$ implies $\lambda \not\bowtie_\mathcal{B} \lambda'$. In view of the assumption on λ, we have proven (a). $\square$

In the proof of the next proposition we denote the transitive closure of a relation $\mathcal{R}$ with $\mathcal{R}^+$, and the reflexive and transitive closure with $\mathcal{R}^\star$. We will use this notation throughout the paper.

Proposition 3.5. *Suppose $\rho_1 \sim \rho_2$. Then for every $\sigma \in$ SCts and every $\mathcal{B}$, $\rho_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma$ implies $\rho_1 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma$.*

Proof. We have to prove the inclusion $\sim \cdot \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \subseteq \dashv\vdash^{\mathcal{B}}_{\text{P2P}}$, and it suffices to show a relation $\mathcal{R}$ such that $\sim \cdot \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \subseteq \mathcal{R}$ and that $\mathcal{R} \subseteq C^{\text{P2P}}(\mathcal{R}, \mathcal{B})$. Let

$$\mathcal{R} = \{\, (\rho', \sigma') \mid \text{ for some } \rho_1, \rho_2, \sigma \in \text{SCts such that } \rho_1 \xrightarrow{\tau}{}^\star \rho', \sigma \xrightarrow{\tau}{}^\star \sigma', \rho_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma, \rho_1 \sim \rho_2 \,\}$$

The relation $\mathcal{R}$ contains by construction the relation $\sim \cdot \dashv\vdash^{\mathcal{B}}_{\text{P2P}}$, and in the rest of the proof we show that $\mathcal{R} \subseteq C^{\text{P2P}}(\mathcal{R}, \mathcal{B})$.

Fix a pair $\rho' \mathrel{\mathcal{R}} \sigma'$, the construction of $\mathcal{R}$ ensures that there exist three contracts $\rho_1, \rho_2, \sigma \in$ SCts that enjoy the following properties

$$\rho_1 \sim \rho_2, \quad \rho_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma, \quad \rho_1 \xrightarrow{\tau}{}^\star \rho', \quad \sigma \xrightarrow{\tau}{}^\star \sigma'$$

Definition 3.3, $\rho_1 \sim \rho_2$, and $\rho_1 \xrightarrow{\tau}{}^\star \rho'$ imply that there exists a ρ'_2 such that $\rho_2 \xrightarrow{\tau}{}^\star \rho'_2$ and $\rho' \sim \rho'_2$. Part ((ii)) of Definition 3.1, $\rho_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma$, and $\rho_2 \xrightarrow{\tau}{}^\star \rho'_2$ let us prove that $\rho'_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma'$.

Definition 3.1 requires us to prove two facts, namely

(i) if $\rho' \parallel \sigma' \xrightarrow{\tau}\!\!\!\!\!/\;_\mathcal{B}$ then $\rho \xrightarrow{\checkmark}, \sigma' \xrightarrow{\checkmark}$
(ii) if $\rho' \parallel \sigma' \xrightarrow{\tau}_\mathcal{B} \rho'' \parallel \sigma''$ then $\rho'' \mathrel{\mathcal{R}} \sigma''$

To prove part ((i)) suppose that $\rho' \parallel \sigma' \xrightarrow{\tau}\!\!\!\!\!/\;_\mathcal{B}$. Since $\rho' \sim \rho'_2$, we apply Lemma 3.4, and obtain that $\rho'_2 \parallel \sigma' \xrightarrow{\tau}\!\!\!\!\!/\;_\mathcal{B}$. Since $\rho'_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma'$, part ((i)) of Definition 3.1 imply that $\rho'_2 \xrightarrow{\checkmark}$ and $\sigma' \xrightarrow{\checkmark}$. Now $\rho' \sim \rho'_2$ and Definition 3.3 ensure that $\rho' \xrightarrow{\checkmark}$, as required.

Now we prove part ((ii)). Suppose that $\rho' \parallel \sigma' \xrightarrow{\tau}_\mathcal{B} \rho'' \parallel \sigma''$, we have to show that $\rho'' \mathrel{\mathcal{R}} \sigma''$. The argument is by case analysis on the rule of Figure 4 used to derive the silent move at hand. If [I-Left] were applied, then $\rho' \xrightarrow{\tau} \rho''$ and $\sigma' = \sigma''$. It follows that $\rho_1 \xrightarrow{\tau}{}^\star \rho''$, thus the construction of $\mathcal{R}$ ensures that $\rho'' \mathrel{\mathcal{R}} \sigma''$. If [I-Right] was applied, we reason in the same manner. Suppose now that [I-Synch] was applied. In this case $\rho' \xrightarrow{\lambda} \rho''$ and $\sigma' \xrightarrow{\lambda'} \sigma''$ for some λ and λ' such that $\lambda \bowtie_\mathcal{B} \lambda'$. To prove that $\rho'' \mathrel{\mathcal{R}} \sigma''$ we exhibit a contract $\hat{\rho}'_2$ which enjoys the following two properties,

$$\rho' \sim \hat{\rho}'_2, \quad \hat{\rho}'_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma'$$

Since $\rho' \sim \rho'_2$ and $\rho' \xrightarrow{\lambda} \rho''$, Definition 3.3 ensures that there exists a contract $\hat{\rho}'_2$ such that $\rho'_2 \xrightarrow{\lambda} \hat{\rho}'_2$ and that $\rho'' \sim \hat{\rho}'_2$. The action performed by ρ'_2 lets us infer the hand-shake $\rho'_2 \parallel \sigma' \xrightarrow{\tau}_\mathcal{B} \hat{\rho}'_2 \parallel \sigma''$. Now $\rho'_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma'$, and part ((ii)) of Definition 3.1 ensure that $\hat{\rho}'_2 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma''$. Observe that $\hat{\rho}'_2$ enjoys the two properties we required above, thus $\rho'' \mathrel{\mathcal{R}} \sigma''$. $\square$

Definition 3.6. [$\mathcal{B}$-peer subcontract preorder]
For σ_1, $\sigma_2 \in \mathsf{SCts}$ let $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ whenever $\rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \sigma_1$ implies $\rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \sigma_2$, for every $\rho \in \mathsf{SCts}$. We use $\sigma_1 =^{\mathcal{B}}_{\mathrm{P2P}} \sigma_2$ to denote the equivalence associated to $\sqsubseteq^{\mathcal{B}}$.

3.2. **Syntactic characterisation.** The parametrised peer subcontract preorder $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ is set based, and quantifies over the result of all peers in $\mathcal{B}$-compliance with σ_1. However, because of the restricted nature of higher-order contracts, Figure 2, it turns out that $\sqsubseteq^{\mathcal{B}}$ can be characterised by the syntactic structure of σ_1 and σ_2, at least for relations $\mathcal{B}$ which satisfy certain minimal conditions.

Let $\mathcal{S} : \mathcal{P}(\mathsf{SCts}^2) \times \mathcal{P}(\mathsf{SCts}^2) \longrightarrow \mathcal{P}(\mathsf{SCts}^2)$ be the functional such 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 $\mathsf{unfold}(\sigma_1) = ?\mathsf{t}_1.\sigma'_1$ then $\mathsf{unfold}(\sigma_2) = ?\mathsf{t}_2.\sigma'_2$ and $\sigma'_1 \mathcal{R} \sigma'_2$ and $\mathsf{t}_1 \leqslant_\mathsf{b} \mathsf{t}_2$
(iii) if $\mathsf{unfold}(\sigma_1) = !\mathsf{t}_1.\sigma'_1$ then $\mathsf{unfold}(\sigma_2) = !\mathsf{t}_2.\sigma'_2$ and $\sigma'_1 \mathcal{R} \sigma'_2$ and $\mathsf{t}_2 \leqslant_\mathsf{b} \mathsf{t}_1$
(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 $\mathsf{unfold}(\sigma_1) = \sum_{i \in I} ?\mathsf{l}_i.\sigma^1_i$ then $\mathsf{unfold}(\sigma_2) = \sum_{j \in J} ?\mathsf{l}_j.\sigma^2_j$ where $I \subseteq J$ and $\sigma^1_i \mathcal{R} \sigma^2_i$ for all $i \in I$
(vii) if $\mathsf{unfold}(\sigma_1) = \bigoplus_{i \in I} !\mathsf{l}_i.\sigma^1_i$ then $\mathsf{unfold}(\sigma_2) = \bigoplus_{j \in J} !\mathsf{l}_j.\sigma^2_j$ where $J \subseteq I$ and $\sigma^1_j \mathcal{R} \sigma^2_j$ for all $j \in J$

Lemma 3.7. *The functional $\mathcal{S}$ is monotone in both arguments:*

(a) *Fix a $\mathcal{B}$. If $\mathcal{R} \subseteq \mathcal{R}'$, then $\mathcal{S}(\mathcal{R}, \mathcal{B}) \subseteq \mathcal{S}(\mathcal{R}', \mathcal{B})$*
(b) *Fix a $\mathcal{R}$. If $\mathcal{B} \subseteq \mathcal{B}'$, then $\mathcal{S}(\mathcal{R}, \mathcal{B}) \subseteq \mathcal{S}(\mathcal{R}, \mathcal{B}')$*

Proof. The proofs of both a) and b) are straightforward. $\qquad\square$

Definition 3.8. [$\mathcal{B}$-syntactic peer preorder]
If $\mathcal{R} \subseteq \mathcal{S}(\mathcal{R}, \mathcal{B})$, then we say that $\mathcal{R}$ is a $\mathcal{B}$-*coinductive peer preorder*. Fix a $\mathcal{B}$. Standard arguments based on part ((a)) of the previous lemma 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 $\leqslant^{\mathcal{B}}$.

One immediate consequence of the syntactic nature of this preorder is that it is preserved by unfolding. This is the first part of the following lemma:

Lemma 3.9.
(1) *For every $\sigma \in \mathsf{SCts}$, $\sigma \leqslant^{\mathcal{B}} \mathsf{unfold}(\sigma) \leqslant^{\mathcal{B}} \sigma$.*
(2) *If $\sigma_1 \leqslant^{\mathcal{B}} \sigma_2$ and σ_2 is stable, then there exist some stable σ'_1 such that $\sigma_1 \xrightarrow{\tau} {}^{\star} \sigma'_1$ and that $\sigma'_1 \leqslant^{\mathcal{B}} \sigma_2$.*

Proof. Part (1) of the lemma follows from the fact that $\rho \, \mathcal{S}(\mathcal{R}, \mathcal{B}) \, \sigma$ if and only if $\mathsf{unfold}(\rho) \, \mathcal{S}(\mathcal{R}, \mathcal{B}) \, \mathsf{unfold}(\sigma)$. This property of $\mathcal{S}(-)$ in turn relies on the fact that unfold is idempotent, that is

$$\mathsf{unfold}(\mathsf{unfold}(\sigma)) = \mathsf{unfold}(\sigma)$$

To prove part (2) suppose $\sigma_1 \leqslant^{\mathcal{B}} \sigma_2$. Part (1) ensures that $\mathsf{unfold}(\sigma_1) \leqslant^{\mathcal{B}} \sigma_2$. If $\mathsf{unfold}(\sigma_1)$ is not stable then it must be an internal sum of the form $\bigoplus_{i \in I} !\mathsf{l}_i.\rho_i$ and since σ_2 is stable it must be of the form $!\mathsf{l}_k.\rho'_k$ for some $k \in I$. The required σ'_1 in this case is $!\mathsf{l}_k.\rho_k$. $\qquad\square$

The reflexivity of $\leqslant^{\mathcal{B}}$ depends tightly on the reflexivity of $\mathcal{B}$.

Lemma 3.10. *If $\mathcal{B}$ is reflexive then $\leqslant^{\mathcal{B}}$ is reflexive.*

Proof. It suffices to prove that the identity relation $\mathcal{I}$ is contained in $\leqslant^{\mathcal{B}}$. To prove this, we show that $\mathcal{I} \subseteq \mathcal{S}(\mathcal{I}, \mathcal{B})$. Fix a pair $\sigma \, \mathcal{I} \, \sigma$. The reasoning is by case analysis on σ, and the only two cases worthwhile involve a higher-order σ. We discuss one such case.

Suppose that $\sigma = !(\sigma^m).\sigma'$. To show that $\sigma \, \mathcal{S}(\mathcal{I}, \mathcal{B}) \, \sigma$, we have to explain why $\sigma' \, \mathcal{I} \, \sigma'$ and $\sigma^m \, \mathcal{B} \, \sigma^m$. The first fact follows form the reflexivity of $\mathcal{I}$, and the second from the reflexivity of the relation $\mathcal{B}$. $\qquad\square$

In the previous lemma the hypothesis of $\mathcal{B}$ being reflexive is not only sufficient, but also necessary for $\leqslant^{\mathcal{B}}$ to be reflexive.

Example 3.11. In this example we prove that if $\mathcal{B}$ is not reflexive then $\leqslant^{\mathcal{B}}$ need not be reflexive.

Let $\sigma = !(1).1$. The empty binary relation $\emptyset$ is not reflexive because $(1, 1)$ is not in $\emptyset$, so we take $\emptyset$ as our candidate $\mathcal{B}$. In turn this implies that $(\sigma, \sigma) \notin \mathcal{S}(\leqslant^\emptyset, \emptyset)$, because 1 does not satisfy the conditions required by case ((iv)) of Definition 3.8. Since $\leqslant^\emptyset = \mathcal{S}(\leqslant^\emptyset, \emptyset)$, it follows that $\sigma \not\leqslant^\emptyset \sigma$.

Our intention is to show that the set-theoretic relation $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ coincides with the syntactically defined relation $\sigma_1 \leqslant^{\mathcal{B}} \sigma_2$, provided $\mathcal{B}$ satisfies some simple properties. In one direction the proof requires the following technical lemma showing that $\sqsubseteq^{\mathcal{B}}$ preserves the ability of contracts to interact.

Lemma 3.12. *Suppose $\rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \sigma_1$ and $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ for some $\mathcal{B}$, where all of σ_1, σ_2, ρ are stable. Then $\rho \parallel \sigma_2 \xrightarrow{\tau}_{\mathcal{B}}$ implies $\rho \parallel \sigma_1 \xrightarrow{\tau}_{\mathcal{B}}$.*

Proof. We know that $\rho \parallel \sigma_2 \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma_2'$ for some pair ρ', σ_2'. Since ρ, σ_2 are stable this reduction must involve interaction between these contracts. That is the derivation of the reduction must end with an application of rule [I-Synch] from Figure 4. The side condition and the premises of the rule ensure that $\rho \xrightarrow{\lambda_1}$ for some λ_1. In turn this move can only be inferred by an application of rule [a-Pre] or [a-Ext] of Figure 3. In both cases one sees that $\rho \xrightarrow{\checkmark}\!\!\!\!\!/$. But by hypothesis $\rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \sigma_1$, so part ((i)) of Definition 3.1 ensures that $\rho \parallel \sigma_1 \xrightarrow{\tau}_{\mathcal{B}}$. $\qquad\square$

Corollary 3.13. *Suppose $\sigma_1 \leqslant^{\mathcal{B}} \sigma_2$ where $\mathcal{B}$ is transitive and $\rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \sigma_1$. Then $\rho \parallel \sigma_2 \xrightarrow{\tau}\!\!\!\!\!/_{\mathcal{B}}$ implies $\rho \xrightarrow{\checkmark}$ and $\sigma_2 \xrightarrow{\checkmark}$*

Proof. We know that ρ and σ_2 are stable, and we let σ_1' be the stable contract guaranteed by the second part Lemma 3.9 such that $\sigma_1 \xrightarrow{\tau}{}^\star \sigma_1'$ and $\sigma_1' \leqslant^{\mathcal{B}} \sigma_2$. Simple properties of compliance ensure that $\rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \sigma_1'$. Therefore by Lemma 3.12 we know that $\rho \xrightarrow{\checkmark}$ and $\sigma_1' \xrightarrow{\checkmark}$. This means that σ_1' is actually 1, so, since $\sigma_1' \leqslant^{\mathcal{B}} \sigma_2$ and σ_2 is stable, σ_2 must also be 1; that is $\sigma_2 \xrightarrow{\checkmark}$ as required. $\square$

Theorem 3.14. *Let $\mathcal{B}$ be a transitive relation on session contracts. Then $\sigma_1 \leqslant^{\mathcal{B}} \sigma_2$ implies $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$.*

Proof. It suffices to show that the following relation is a $\mathcal{B}$-coinductive peer compliance,

$$\mathcal{R} = \{ (\rho, \sigma_2) \mid \rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \sigma_1, \, \sigma_1 \leqslant^{\mathcal{B}} \sigma_2, \text{ for some } \sigma_1 \in \mathsf{SCts} \}$$

This requires establishing two properties.

(i) If $\rho \parallel \sigma_2 \xrightarrow{\tau}\!\!\!\!\!/_{\mathcal{B}}$ whenever $\rho \, \mathcal{R} \, \sigma_2$ then $\rho \xrightarrow{\checkmark}$ and $\sigma_2 \xrightarrow{\checkmark}$. This follows from Corollary 3.13.

(ii) Suppose $\rho \parallel \sigma_2 \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma_2'$, where $\rho \mathcal{R} \sigma_2$. We have to prove that $\rho' \mathcal{R} \sigma_2'$.

Because $\rho \mathcal{R} \sigma_2$ we know there exists some σ_1 such that $\rho \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1$ and $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$. We have to find some σ_1' such that $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1'$ and $\sigma_1' \preccurlyeq^{\mathcal{B}} \sigma_2'$. The nature of σ_1' depends on the form of the reduction $\rho \parallel \sigma_2 \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma_2'$.

If this is a silent move by ρ to ρ' we can take σ_1' to be σ_1 itself, since $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1$. If it is a silent move by σ_2 then there are two cases. If the move is inferred using the rule [A-UNFOLD] from Figure 3 then $\mathsf{unfold}(\sigma_2) = \mathsf{unfold}(\sigma_2')$, which ensures that $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2'$.

The only other possible way for σ_2 to do a silent move is by an application of rule [A-INT]. Here σ_2 has the form $\bigoplus_{i \in I} !1_i.\sigma_i^2$ and σ_2' must be $!1_k.\sigma_k^2$ for some $k \in I$. In this case again we can take σ_1' to be σ_1, since $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2'$.

If the reduction is due to an interaction between ρ and σ_2 then there are six cases to discuss. The argument for each of them is similar, so we discuss only one case involving higher-order communication. Suppose that $\rho = ?(\rho^m).\rho'$. Since σ_2 engages in a communication with ρ it must be the case that $\sigma_2 = !(\sigma_2^m).\sigma_2'$ with $\sigma_2^m \mathcal{B} \rho^m$. Since $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$, it follows that $\mathsf{unfold}(\sigma_1) = !(\sigma_1^m).\sigma_1'$ with $\sigma_1^m \mathcal{B} \sigma_2^m$ and $\sigma_1' \preccurlyeq^{\mathcal{B}} \sigma_2'$. It remains to show that $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1'$. The transitivity of $\mathcal{B}$ ensures that $\sigma_1^m \mathcal{B} \rho^m$, so we can infer $\rho \parallel \sigma_1 \xrightarrow{\tau}{}^{\star}_{\mathcal{B}} \rho' \parallel \sigma_1'$, which implies that $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1'$. $\square$

Example 3.15. We show that if $\mathcal{B}$ is not a transitive relation, then $\preccurlyeq^{\mathcal{B}}$ need not be contained in $\sqsubseteq^{\mathcal{B}}$, that is $\preccurlyeq^{\mathcal{B}} \not\subseteq \sqsubseteq^{\mathcal{B}}$.

Let $\mathcal{B} = \{(1, !1.!1.1), (!1.!1.1, !1.1)\}$. This relation is not transitive, because $1 \mathcal{B} !1.!1.1$ and $!1.!1.1 \mathcal{B} !1.1$, while $(1, !1.1) \notin \mathcal{B}$.

Let $\sigma_1 = !(!1.!1.1).1$ and let $\sigma_2 = !(1).1$. We show that $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$. The witness of this fact is the relation $\mathcal{R} = \{(\sigma_1, \sigma_2), (1, 1)\}$. We are required to prove that $\mathcal{R} \subseteq \mathcal{S}(\mathcal{R}, \mathcal{B})$. This amounts in showing that a) $1 \mathcal{S}(\mathcal{R}, \mathcal{B}) 1$, and b) $\sigma_1 \mathcal{S}(\mathcal{R}, \mathcal{B}) \sigma_2$. Point a) is true thanks to case ((i)) of Definition 3.8, and point b) follows from case ((iv)) of the same definition.

Now we prove that $\sigma_1 \not\sqsubseteq^{\mathcal{B}} \sigma_2$. We have to exhibit a session contract ρ, such that $\rho \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1$, and $\rho \not\dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_2$. Let $\rho = ?(!1.1).1$. To see why $\rho \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1$, note that the relation $\{(\rho, \sigma_1), (1, 1)\}$ is a $\mathcal{B}$-coinductive mutual compliance.

To conclude the example, we have to prove that $\rho \not\dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_2$. The witness that $\mathcal{B}$ is not transitive is the fact that $(1, !1.1) \notin \mathcal{B}$. This implies that $!(1) \not\bowtie_{\mathcal{B}} ?(!1.1)$, and in turn that $\rho \parallel \sigma_2 \xrightarrow{\tau}\!\!\!\!\!\!/\,\,_{\mathcal{B}}$. Since $\rho \xrightarrow{\checkmark}\!\!\!\!\!/\,\,$, it follows that $\rho \not\dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_2$.

The converse to Theorem 3.14 relies on following property of session contracts, whose proof is relegated to Section 5; see Theorem 5.17.

Theorem 3.16. *Let $\mathcal{B}$ be a preorder on session contracts. For every session contract ρ there exists a session contract $\mathsf{prdual}(\rho)$ such that $\rho \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \mathsf{prdual}(\rho)$.*

Theorem 3.17. *Let $\mathcal{B}$ be a preorder on session contracts. Then $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ implies $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$.*

Proof. Since $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ implies that $\mathsf{unfold}(\sigma_1) \sqsubseteq^{\mathcal{B}} \mathsf{unfold}(\sigma_2)$, it is enough to prove that $\mathcal{R}$ is a $\mathcal{B}$-coinductive peer preorder, $\mathcal{R} \subseteq \mathcal{S}(\mathcal{R}, \mathcal{B})$, where $\mathcal{R}$ is given by

$$\mathcal{R} = \{(\sigma_1, \sigma_2) \mid \mathsf{unfold}(\sigma_1) \sqsubseteq^{\mathcal{B}} \mathsf{unfold}(\sigma_2)\}$$

Pick a pair $\sigma_1 \mathcal{R} \sigma_2$. To show that $\sigma_1 \mathcal{S}(\mathcal{R}, \mathcal{B}) \sigma_2$ we reason by case analysis on the unfoldings of these contracts; the argument for many cases are similar, so we only discuss two cases.

- Suppose that $\text{unfold}(\sigma_1) = 1$. The relation $\{(1, 1)\}$ is a coinductive $\mathcal{B}$-mutual compliance, so $1 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{unfold}(\sigma_1)$. As $\text{unfold}(\sigma_1) \sqsubseteq^{\mathcal{B}} \text{unfold}(\sigma_2)$, it follows that $1 \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{unfold}(\sigma_2)$. A simple argument, based on the possible structure of σ_2, will show that $\text{unfold}(\sigma_2)$ is stable, from which it follows that $1 \parallel \text{unfold}(\sigma_2) \overset{\tau}{\nrightarrow}$. Compliance now ensures that $\text{unfold}(\sigma_2) \overset{\checkmark}{\longrightarrow}$. Because of the restrictive syntax for contracts this is only possible if $\text{unfold}(\sigma_2)$ is actually 1. It follows that $\sigma_1 \; \mathcal{S}(\mathcal{R}, \mathcal{B}) \; \sigma_2$.

- Suppose that $\text{unfold}(\sigma_1) = !(\sigma_1^m).\sigma_1'$. We have to prove the equality

$$\text{unfold}(\sigma_2) = !(\sigma_2^m).\sigma_2'$$

where $\sigma_2^m \; \mathcal{B} \; \sigma_1^m$, and $\sigma_1' \; \mathcal{R} \; \sigma_2'$.

Theorem 3.16 and the hypothesis on $\mathcal{B}$ ensures the existence of some contract $\text{prdual}(\sigma_1')$ such that $\sigma_1' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{prdual}(\sigma_1')$. Let $\rho = ?(\sigma_1^m).\text{prdual}(\sigma_1')$ and let

$$\mathcal{R}' = \{(\rho, \text{unfold}(\sigma_1))\} \cup \dashv\vdash^{\mathcal{B}}_{\text{P2P}}$$

the relation $\mathcal{R}'$ is a $\mathcal{B}$-coinductive peer compliance. This is the case because the preorder $\mathcal{B}$ is reflexive, thus there exists the derivation

$$\frac{\rho \overset{?(\sigma_1^m)}{\longrightarrow} \text{prdual}(\sigma_1') \quad \text{unfold}(\sigma_1) \overset{!(\sigma_1^m)}{\longrightarrow} \sigma_1'}{\rho \parallel \text{unfold}(\sigma_1) \overset{\tau}{\longrightarrow}_{\mathcal{B}} \text{prdual}(\sigma_1') \parallel \sigma_1'} \; ?(\sigma_1^m) \bowtie_{\mathcal{B}} !(\sigma_1^m)$$

and because, thanks to the symmetry of $\dashv\vdash^{\mathcal{B}}_{\text{P2P}}$, $\text{prdual}(\sigma_1') \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1'$.

It follows that $\rho \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{unfold}(\sigma_1)$, and since $\text{unfold}(\sigma_1) \sqsubseteq^{\mathcal{B}} \text{unfold}(\sigma_2)$, we obtain immediately that $\rho \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{unfold}(\sigma_2)$.

Since $\rho \overset{\checkmark}{\nrightarrow}$, the composition $\rho \parallel \text{unfold}(\sigma_2)$ performs a silent move. The syntax of session contracts ensures that $\text{unfold}(\sigma_2)$ cannot be an internal sum, and therefore $\text{unfold}(\sigma_2)$ has to interact with ρ. The definition of $\bowtie_{\mathcal{B}}$ ensures that $\text{unfold}(\sigma_2)$ has to have the form $!(\sigma_2^m).\sigma_2'$ where $\sigma_2^m \; \mathcal{B} \; \sigma_1^m$, our first requirement. Moreover $\rho \parallel \text{unfold}(\sigma_2) \overset{\tau}{\longrightarrow}_{\mathcal{B}} \rho' \parallel \sigma_2'$ from which $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_2'$ follows.

To show the second requirement, $\sigma_1' \; \mathcal{R} \; \sigma_2'$, let ρ' be any contract satisfying the condition $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{unfold}(\sigma_1')$; we have to prove that $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{unfold}(\sigma_2')$. Note that because of Lemma 3.9(1) we can also assume that $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_1'$. We repeat the above argument to establish that $?(\sigma_1^m).\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{unfold}(\sigma_2)$, and since

$$?(\sigma_1^m).\rho' \parallel \text{unfold}(\sigma_2) \overset{\tau}{\longrightarrow}_{\mathcal{B}} \rho' \parallel \sigma_2'$$

we know that $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \sigma_2$. The required $\rho' \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \text{unfold}(\sigma_2')$ now follows by another application of Lemma 3.9(1). $\qquad \square$

Example 3.18. We show that if $\mathcal{B}$ is not a reflexive relation, then $\sqsubseteq^{\mathcal{B}}$ needs not be contained in $\preccurlyeq^{\mathcal{B}}$: $\sqsubseteq^{\mathcal{B}} \nsubseteq \preccurlyeq^{\mathcal{B}}$.

Recall Example 3.11, and the session contract we employed there, $\sigma = !(1).1$. We know that $\sigma \sqsubseteq^{\emptyset} \sigma$, because no peer can interact with $!(1)$, so no peer complies with σ. However in Example 3.11 we have proven that $\sigma \npreccurlyeq^{\emptyset} \sigma$.

Corollary 3.19. *For any preorder $\mathcal{B}$ over session contracts, $\sigma_1 \preccurlyeq^{\mathcal{B}} \sigma_2$ if and only if $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$.*

Proof. Follows immediately from Theorem 3.14 and Theorem 3.17. $\qquad \square$

4. Modelling session types

In our treatment session types and contracts are obviously just syntactic variations on each other. We formalise the relationship between them as a function which maps session types from STyp to session contracts from SCts,

$$\mathcal{M} : L_{\mathsf{STyp}} \longrightarrow L_{\mathsf{SCts}}$$

We then 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 $\mathcal{B}$.

The interpretation of types into contracts is defined by the following syntactic translation:

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

The function $\mathcal{M}^{-1} : L_{\mathsf{SCts}} \longrightarrow L_{\mathsf{STyp}}$ is the obvious inverse of $\mathcal{M}$, for instance

$$\mathcal{M}^{-1}(!\mathsf{t}.\sigma) = !(\mathsf{t}).\mathcal{M}^{-1}(\sigma),$$

and we omit its definition.

Because of the syntactic nature of $\mathcal{M}$ and $\mathcal{M}^{-1}$ the following properties are easy to establish.

Lemma 4.1. *For every $S, T \in L_{\mathsf{STyp}}$ and $\rho, \sigma \in L_{\mathsf{SCts}}$,*

a) $\mathcal{M}(S[X \mapsto T]) = (\mathcal{M}(S))[\mathcal{M}(X) \mapsto \mathcal{M}(T)]$
b) $\mathcal{M}^{-1}(\rho[x \mapsto \sigma]) = (\mathcal{M}^{-1}(\rho))[\mathcal{M}^{-1}(x) \mapsto \mathcal{M}^{-1}(\sigma)]$
c) $\mathsf{unfold}(\mathcal{M}(T)) = \mathcal{M}(\mathsf{unfold}(T))$
d) $\mathsf{unfold}(\mathcal{M}^{-1}(\sigma)) = \mathcal{M}^{-1}(\mathsf{unfold}(\sigma))$
e) $\mathsf{unfold}(\mathcal{M}^{-1}(\sigma)) = T$ *iff* $\mathsf{unfold}(\sigma) = \mathcal{M}(T)$

Proof. Part (a)) and part (b)) are proven by structural induction, respectively on S and ρ. Part (c)) uses rule induction on $\mathsf{unfold}(T)$ together with an application of part (a)). Part (d)) is proven by rule induction on $\mathsf{unfold}(\sigma)$, and uses part (b)). Part (e)) is a consequence of (c)) and (d)). $\qquad\square$

In order to find the appropriate $\mathcal{B}$ that captures the subtyping relation $S \preccurlyeq T$ via the interpretation $\mathcal{M}$, 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.

Lemma 4.2. $(\mathcal{P}re, \subseteq)$ *is a complete lattice.*

Proof. We have to show that all the subsets of $\mathcal{P}re$ have infimum and supremum. Let $X \subseteq \mathcal{P}re$. The infimum of X is defined as the intersection of the elements of X, that is $\bigsqcap X = \bigcap \{ \mathcal{B} \mid \mathcal{B} \in X \}$. The supremum of X is defined as the transitive closure of the union of the elements of X, that is $\bigsqcup X = (\bigcup \{ \mathcal{B} \mid \mathcal{B} \in X \})^+$. It is routine work to check that $\bigsqcap X \subseteq \mathcal{B}$ and $\mathcal{B} \subseteq \bigsqcup X$ for every $\mathcal{B} \in X$. $\square$

Let $\mathcal{F}^{\mathrm{P2P}} : \mathcal{P}re \longrightarrow \mathcal{P}re$ be defined by letting $\mathcal{F}^{\mathrm{P2P}}(\mathcal{B})$ be $\sqsubseteq^{\mathcal{B}}$.

Proposition 4.3. *$\mathcal{F}^{\mathrm{P2P}}$ is a monotone endofunction.*

Proof. A priori there is no simple direct argument to show, using Definition 3.6, that $\mathcal{F}^{\mathrm{P2P}}$ is monotonic. But the result is now a direct corollary of part ((b)) of Lemma 3.7, and of Corollary 3.19. $\square$

Definition 4.4. [Peer subcontract preorder]
We use $\sqsubseteq$ to denote $\nu X.\mathcal{F}^{\mathrm{P2P}}(X)$, the greatest fixed point of the monotone function $\mathcal{F}^{\mathrm{P2P}}$. The existence of this fixed point is guaranteed by Proposition 4.3. We refer to $\sqsubseteq$ as the *Peer subcontract preorder*.
    We also let $\approx$ denote the *Peer equivalence* generated by $\sqsubseteq$ in the obvious way.

The properties of $\sqsubseteq$ alluded to in (1.7) of the Introduction are now easy to establish.

Proposition 4.5. *$\sqsubseteq$ is the largest preorder $\mathcal{B}$ over SCts satisfying: $\sigma_1 \ \mathcal{B} \ \sigma_2$ if and only if $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$.*

Proof. A direct consequence of the fact that $\sqsubseteq$ is the greatest fixed point of $\mathcal{F}^{\mathrm{P2P}}$. $\square$

The proof that $\sqsubseteq$ provides a fully-abstract model of subtyping $\leqslant$ on session types , relies on another characterisation, which in turn uses a standard result from lattice theory [AN01, pag. 19].

Lemma 4.6. *[Golden lemma]*
Let L be a complete lattice and $f : L \times L \longrightarrow L$ an endofunction monotone in both arguments. Then $\nu y.\nu x.f(x,y) = \nu x.f(x,x)$.[1]

Lemma 4.7. *$\sqsubseteq = \nu X.\mathcal{S}(X, X)$.*

Proof. By definition $\sqsubseteq = \nu X.\mathcal{F}^{\mathrm{P2P}}(X) = \nu X.\sqsubseteq^X$. But by Corollary 3.19 the preorder $\sqsubseteq^{\mathcal{B}}$ coincides with the relation $\leqslant^{\mathcal{B}}$ for any preorder $\mathcal{B}$. Since $\mathcal{F}^{\mathrm{P2P}}$ is a function over $\mathcal{P}re$, we have $\sqsubseteq = \nu Y.\leqslant^Y$. Definition 3.8 lets us expand $\leqslant^{\mathcal{B}}$, thereby obtaining the equality $\sqsubseteq = \nu Y.\nu X.\mathcal{S}(X, Y)$. The result is now a consequence of the Golden lemma. $\square$

To obtain the full-abstraction result, we show how the prefixed points of the functionals $\mathcal{S}$ and $\mathcal{F}^{\leqslant}$ are related via $\mathcal{M}$.

Lemma 4.8. *Fix a relation $\mathcal{B}$ such that $\mathcal{B} \subseteq \mathcal{S}(\mathcal{B}, \mathcal{B})$ and let*
$$\mathcal{T} = \{ (\mathcal{M}^{-1}(\sigma_1), \mathcal{M}^{-1}(\sigma_2)) \mid \sigma_1 \ \mathcal{B} \ \sigma_2 \}$$
Then $\mathcal{T} \subseteq \mathcal{F}^{\leqslant}(\mathcal{T})$.

Proof. (Outline) Fix a pair $S_1 \ \mathcal{T} \ S_2$. These types are the images via $\mathcal{M}^{-1}$ of two session contracts, respectively σ_1 and σ_2, such that $\sigma_1 \ \mathcal{B} \ \sigma_2$.
    The proof proceeds by a case analysis on the structure of $\mathsf{unfold}(\mathcal{M}^{-1}(\sigma_1))$; we give the details of two cases.
- Suppose $\mathsf{unfold}(\mathcal{M}^{-1}(\sigma_1)) = \textsc{end}$. According to Definition 2.3 we then have to show that $\mathsf{unfold}(\mathcal{M}^{-1}(\sigma_2)) = \textsc{end}$. Because of Lemma 4.1(e)) we know that $\mathsf{unfold}(\sigma_1) = 1$; case ((i)) of Definition 3.8 ensures that $\mathsf{unfold}(\sigma_2) = 1$, and Lemma 4.1 therefore implies the syntactic equality $\mathsf{unfold}(\mathcal{M}^{-1}(\sigma_2)) = \textsc{end}$.

---

[1]The result proven in [AN01] is more general; it pertains to both least and greatest fixed points of endofunctions.

- Suppose $\mathsf{unfold}(\mathcal{M}^{-1}(\sigma_1)) = \,!(T_1).S_1$. We are required to prove that

$$\mathsf{unfold}(\mathcal{M}^{-1}(\sigma_2)) = \,!(T_2).S_2 \tag{4.1}$$

for some T_2 and S_2 such that $T_2 \; \mathcal{T} \; T_1$ and $S_1 \; \mathcal{T} \; S_2$.

Lemma 4.1(e)) ensures that $\mathsf{unfold}(\sigma_1) = \,!(\mathcal{M}(T_1)).\mathcal{M}(S_1)$. Since we know that $\sigma_1 \; \mathcal{B} \; \sigma_2$, the hypothesis that $\mathcal{B} \subseteq \mathcal{S}(\mathcal{B}, \mathcal{B})$ and Definition 3.8 imply the equality

$$\mathsf{unfold}(\sigma_2) = \,!(\sigma_2^m).\sigma_2'$$

with σ_2^m such that $\sigma_2^m \; \mathcal{B} \; \mathcal{M}(T_1)$ and some σ_2' such that $\mathcal{M}(S_1) \; \mathcal{B} \; \sigma_2'$. The construction of $\mathcal{T}$ implies that

$$\mathcal{M}^{-1}(\sigma_2^m) \; \mathcal{T} \; T_1, \quad S_1 \; \mathcal{T} \; \mathcal{M}^{-1}(\sigma_2')$$

Because of $\sigma_2 = \mathcal{M}(S_2)$, Lemma 4.1 implies the syntactic equality $\mathsf{unfold}(\sigma_2) = \mathcal{M}(\mathsf{unfold}(S_2))$, thus $\mathsf{unfold}(S_2) = \,!(\mathcal{M}^{-1}(\sigma_2^m)).\mathcal{M}^{-1}(\sigma_2')$. This ensures that (4.1) above is satisfied.

The proof for the remaining cases is similar to the argument already shown, and left to the reader. $\square$

Lemma 4.9. *Let $\mathcal{T}$ be a type simulation, and*

$$\mathcal{B} = \{\, (\mathcal{M}(S), \; \mathcal{M}(T)) \mid S \; \mathcal{T} \; T \,\}$$

Then $\mathcal{B} \subseteq \mathcal{S}(\mathcal{B}, \mathcal{B})$.

Proof. (Outline) Suppose $\sigma_1 \; \mathcal{B} \; \sigma_2$. By construction $\sigma_1 = \mathcal{M}(S_1)$ and $\sigma_2 = \mathcal{M}(S_2)$ for some S_1 and S_2 related by $\mathcal{T}$. The proof is a case analysis on $\mathsf{unfold}(\sigma_1)$.

- If $\mathsf{unfold}(\sigma_1) = 1$ we have to prove that $\mathsf{unfold}(\sigma_2) = 1$. An application of Lemma 4.1(e)) shows that $\mathsf{unfold}(S_1) = \textsc{end}$. The hypothesis that $\mathcal{T}$ is a type simulation ensures that $\mathsf{unfold}(S_1) = \textsc{end}$, so another application of Lemma 4.1 leads to $\mathsf{unfold}(\sigma_2) = 1$.
- If $\mathsf{unfold}(\sigma_1) = \,?(\sigma_1^m).\sigma_1'$ we have to show that

$$\mathsf{unfold}(\sigma_2) = \,?(\sigma_2^m).\sigma_2'$$

with $\sigma_1^m \; \mathcal{B} \; \sigma_2^m$ and $\sigma_1' \; \mathcal{B} \; \sigma_2'$. We apply Lemma 4.1(e)) and obtain the equality

$$\mathsf{unfold}(S_1) = \,?(\mathcal{M}^{-1}(\sigma_1^m)).\mathcal{M}^{-1}(\sigma_1')$$

By hypothesis the relation $\mathcal{T}$ is a type simulation, so $S_1 \; \mathcal{T} \; S_2$ let us deduce that

$$\mathsf{unfold}(S_2) = \,?(T_2).S_2'$$

with

$$\mathcal{M}^{-1}(\sigma_1^m) \; \mathcal{T} \; T_2, \quad \mathcal{M}^{-1}(\sigma_1') \; \mathcal{T} \; S_2'$$

This implies that

$$\sigma_1^m \; \mathcal{B} \; \mathcal{M}(T_2), \quad \sigma_1' \; \mathcal{B} \; \mathcal{M}(S_2')$$

Lemma 4.1, $\mathsf{unfold}(S_2) = \,?(T_2).S_2'$, and $\sigma_2 = \mathcal{M}(S_2)$ ensure the equality

$$\mathsf{unfold}(\sigma_2) = \,?(\mathcal{M}(T_2)).\mathcal{M}(S_2').$$

The other cases are analogous and left to the reader. $\square$

The last two results make the proof of full-abstraction straightforward.

Theorem 4.10. *[Full-abstraction]*
For every $T, S \in \mathsf{STyp}$, $S \leqslant T$ if and only if $\mathcal{M}(S) \sqsubseteq \mathcal{M}(T)$.

Proof. Suppose that $S \leqslant T$. Lemma 4.9 implies that the relation

$$\mathcal{B} = \{ (\mathcal{M}(S), \mathcal{M}(T)) \mid S \leqslant T \}$$

is contained in $\mathcal{S}(\mathcal{B}, \mathcal{B})$, thus $\mathcal{B} \subseteq \nu X.\mathcal{S}(X, X)$. Lemma 4.7 implies that $\mathcal{B} \subseteq \sqsubseteq$. It follows that $\mathcal{M}(S) \sqsubseteq \mathcal{M}(T)$.

Suppose that $\mathcal{M}(S) \sqsubseteq \mathcal{M}(T)$. Note that $\sqsubseteq \subseteq \mathcal{S}(\sqsubseteq, \sqsubseteq)$. Lemma 4.8 implies that the relation

$$\mathcal{T} = \{ (\mathcal{M}^{-1}(\rho), \mathcal{M}^{-1}(\sigma)) \mid \rho \sqsubseteq \sigma \}$$

is a type simulation, so $\mathcal{T} \subseteq \leqslant$. Since $S \, \mathcal{T} \, T$, it follows that $S \leqslant T$. $\qquad\square$

Full-abstraction has two immediate consequence. The first is a result on the decidability of $\sqsubseteq$.

Proposition 4.11. *If $\leqslant_{\mathsf{b}}$ is decidable, then relation $\sqsubseteq$ is decidable.*

Proof. First we describe the an algorithm to decide $\leqslant$. In [GH05, Figure 11, Lemma 10, Corollary 2] an algorithm is presented, which decides $\leqslant$ but for a language of types with no input/output of base types. Adding the following two rules to the ones in Figure 11 of that paper we obtain an algorithmic subtyping relation $\leqslant$, that works also for types with input/output of base types.

$$\frac{\Sigma \vdash S_1 \leqslant S_2}{\Sigma \vdash ?(\mathsf{t}_1).S_2 \leqslant ?(\mathsf{t}_2).S_2} \; \mathsf{t}_1 \leqslant_{\mathsf{b}} \mathsf{t}_2$$

$$\frac{\Sigma \vdash S_1 \leqslant S_2}{\Sigma \vdash !(\mathsf{t}_1).S_2 \leqslant !(\mathsf{t}_2).S_2} \; \mathsf{t}_2 \leqslant_{\mathsf{b}} \mathsf{t}_1$$

Thanks to the hypothesis that $\leqslant_{\mathsf{b}}$ is decidable, Lemma 10, Corollary 2 of [GH05] are true also for $\leqslant$ and $\leqslant$, that is for every session type S_1 and S_2

  i) The algorithmic subtyping $\vdash S_1 \leqslant S_2$ terminates
 ii) $\vdash S_1 \leqslant S_2$ if and only if $S_1 \leqslant S_2$

Now we show how to decide whether two session contracts σ_1 and σ_2 are in the relation $\sqsubseteq$.

1) Let $S_1 = \mathcal{M}^{-1}(\sigma_1)$ and $S_2 = \mathcal{M}^{-1}(\sigma_2)$. The applications of the function $\mathcal{M}^{-1}$ terminates because $\mathcal{M}^{-1}$ is defined inductively,
2) apply the algorithmic subtyping to decide whether $\vdash S_1 \leqslant S_2$. Part (i)) above ensures that the algorithm terminates,
3) Part (ii)) above ensures that the algorithm has decided whether $S_1 \leqslant S_2$,
4) Theorem 4.10 now implies that if $S_1 \leqslant S_2$ then $\sigma_1 \sqsubseteq \sigma_2$, and if $S_1 \nleqslant S_2$ then $\sigma_1 \not\sqsubseteq \sigma_2$. $\quad\square$

The second immediate consequence of Theorem 4.10 is an explanation of type equivalence. *Type equivalence*, denoted $=_{\mathsf{eq}}$, is the equivalence generated by the subtyping, so that

$$T =_{\mathsf{eq}} S \text{ whenever } T \leqslant S \text{ and } S \leqslant T \tag{4.2}$$

The explanation of $=_{\mathsf{eq}}$ is alternative to the standard one based on tree models of types [BH98].
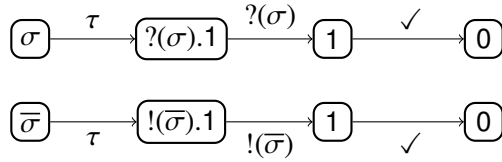
Proposition 4.12. *[Full-abstraction type equivalence]*
For every $T, S \in \mathsf{STyp}$, $S =_{\mathsf{eq}} T$ if and only if $\mathcal{M}(S) \approx \mathcal{M}(T)$.

Proof. A direct consequence of the definitions of $=_{\mathsf{eq}}$, $\approx$ and of Theorem 4.10. $\qquad\square$

$$\overline{\text{END}} = \text{END}, \quad \overline{X} = X, \quad \overline{\mu X.S} = \mu \overline{X}.\overline{S}$$

$$\overline{?(M).S} = !(M).\overline{S}, \quad \overline{!(M).S} = ?(M).\overline{S}$$

$$\overline{\&\langle 1_1 \colon S_1, \ldots, 1_n \colon S_n \rangle} = \oplus\langle 1_1 \colon \overline{S_1}, \ldots, 1_n \colon \overline{S_n} \rangle$$

$$\overline{\oplus\langle 1_1 \colon S_1, \ldots, 1_n \colon S_n \rangle} = \&\langle 1_1 \colon \overline{S_1}, \ldots, 1_n \colon \overline{S_n} \rangle$$

Figure 5: Standard syntactic definition of dual session types

If $\mathcal{B}$ is reasonable, then $!(\overline{\sigma}) \not\Vdash_{\mathcal{B}} ?(\sigma)$, and so σ and $\overline{\sigma}$ are not in $\mathcal{B}$-mutual compliance.

Figure 6: The behaviours of the contracts ρ and σ of Example 5.2

5. Complements of Contracts

The converse to Theorem 3.17 relies on the existence for every session contract σ of a "complementary" session contract $\mathsf{prdual}(\sigma)$ that is in $\mathcal{B}$-peer compliance with σ, at least for $\mathcal{B}$s that satisfy certain minimal conditions. The well-known *syntactic duality* of session types, discussed in the Introduction and defined inductively in Figure 5, is an obvious candidate. It is defined for the language of session contracts L_{SCts} by structural induction in Figure 5; we are primarily interested in it as applied to session contracts SCts. Intuitively to obtain the dual of a contact σ, denoted $\overline{\sigma}$,

- every internal choice is transformed into an external choice
- every external choice is transformed into an internal choice
- inputs are turned into outputs, and outputs into inputs.

But it should be emphasised that in the transformation from σ to $\overline{\sigma}$ all messages are left unchanged.

Unfortunately, as we will see in the following example, this standard duality transformation does not satisfy our requirements for complementary contracts. First a definition.

Definition 5.1. We say that $\mathcal{B}$ is *reasonable* whenever $\sigma_1 \, \mathcal{B} \, \sigma_2$ implies

  i) $\mathsf{unfold}(\sigma_1) \, \mathcal{B} \, \mathsf{unfold}(\sigma_2)$
 ii) if $\sigma_1 \xrightarrow{\lambda_1}$ and $\sigma_2 \xrightarrow{\lambda_2}$ then λ_1 and λ_2 are *both* input actions or output actions.
iii) if $\sigma_1 \xrightarrow{?(\sigma_1^m)} \sigma_1'$ and $\sigma_2 \xrightarrow{?(\sigma_2^m)} \sigma_2'$ then $\sigma_1^m \, \mathcal{B} \, \sigma_2^m$ and $\sigma_1' \, \mathcal{B} \, \sigma_2'$ $\square$

For instance, if $!(1).1 \, \mathcal{B} \, ?(1).1$ for some $\mathcal{B}$, then $\mathcal{B}$ is not reasonable.

The family of reasonable relations is not arbitrary. Theorem 3.17 implies that the $\sqsubseteq^{\mathcal{B}}$ for every preorder $\mathcal{B}$ is reasonable, thus reasonable relations are an over-approximation of the behavioural preorders that we are concerned with.

Example 5.2. In general it is not true that σ complies with its dual $\overline{\sigma}$; if $\mathcal{B}$ is reasonable then we can find a contract σ such that $\sigma \not\Vdash_{\mathsf{P2P}}^{\mathcal{B}} \overline{\sigma}$.

For example take σ to be $\mu x.?(x).1$; here $\overline{\sigma}$ is $\mu x.!(x).1$. The behaviour of these contracts is depicted in Figure 6. Since $\mathsf{unfold}(\sigma)$ performs inputs, while $\mathsf{unfold}(\overline{\sigma})$ performs outputs, and $\mathcal{B}$ is reasonable, one sees that $(\mathsf{unfold}(\overline{\sigma}), \mathsf{unfold}(\sigma)) \notin \mathcal{B}$, and in turn $(\overline{\sigma}, \sigma) \notin \mathcal{B}$. This implies that $!(\overline{\sigma}) \not\leftleftarrows_{\mathcal{B}} ?(\sigma)$, and so $\sigma \parallel \overline{\sigma} \xrightarrow{\tau}_{\mathcal{B}} \mathsf{unfold}(\sigma) \parallel \mathsf{unfold}(\overline{\sigma}) \not\xrightarrow{\tau}_{\mathcal{B}}$. But this means that $\sigma \not\dashv^{\mathcal{B}}_{\mathsf{P2P}} \overline{\sigma}$ because $\mathsf{unfold}(\sigma)$ does not perform $\checkmark$.

5.1. **Message-closed contracts.**
In this section we give a restriction on contracts which ensures that they do indeed comply with their duals. The essential idea is that terms used as messages should have no free occurrences of recursion variables.

Definition 5.3. [Message-closed]
For any $\sigma \in L_{\mathsf{SCts}}$ we say that it is *message-closed*, or *m-closed*, whenever a sub-term of the form $?(\sigma^m).\sigma'$ or $!(\sigma^m).\sigma'$ occurs in the main body of σ then σ^m is a closed contract, that is, is in SCts. More formally:

(1) The terms 1 and x are *m-closed*
(2) The term $\mu x.\sigma'$ is *m-closed* if σ' is *m-closed*.
(3) The terms $!(\sigma^m).\sigma'$ and $?(\sigma^m).\sigma'$ are *m-closed* if σ' is *m-closed* and σ^m is closed.
(4) The terms $\sum_{i \in I} ?1_i.\sigma_i$ and $\bigoplus_{i \in I} !1_i.\sigma_i$ are *m-closed* if all σ_i are *m-closed*.

It is important to note the *m-closed* is quite a strong condition; if σ is closed then it does not automatically follow that it is *m-closed*. As a counterexample we can take the contract used in Example 5.2, $\mu x.?(x).1$.

The crucial property of *m-closed* terms is that the dual function $\overline{}$ is preserved by substitutions. This is expressed in the following lemma where we use $\overline{s}$ to denote the substitution which maps each variable X to $\overline{s(X)}$.

Lemma 5.4.

(i) *Suppose $\sigma \in L_{\mathsf{SCts}}$ is* m-closed. *Then*
    (a) $\overline{(\sigma \mathbf{s})} = (\overline{\sigma})\overline{\mathbf{s}}$
    (b) $\overline{\sigma}$ *is* m-closed
    (c) *if* $\mathbf{s}(x)$ *is* m-closed *for every* $x \in \mathsf{dom}(\mathbf{s})$ *then* $\sigma \mathbf{s}$ *is also* m-closed.
(ii) *For every* m-closed *$\sigma \in \mathsf{SCts}$ and $\mu \in \mathsf{Act}_{\tau \checkmark}$, $\sigma \xrightarrow{\mu} \sigma'$ implies σ' is also* m-closed.

Proof. Part (i) is proved by structural induction on σ. Part (ii) uses rule induction on the judgements $\sigma \xrightarrow{\mu} \sigma'$; the case when σ has the form $\mu x.\sigma_1$ relies on Part (i) (c). $\qquad\square$

The requirement that σ be *m-closed* in Lemma 5.4 is essential. Again the contract $\sigma = \mu x.?(x).1$, used in Example 5.2, provides a counterexample. Let $\mathbf{s}$ be the substitution $[x \mapsto \sigma]$ and let σ' be the body of the recursive definition, $?(x).1$, which is not *m-closed*. Then $\overline{(\sigma' \mathbf{s})}$ is the contract $!(\mu x.?(x).1).1$ whereas, since $\overline{\sigma} = \mu x.!(x).1$, $(\overline{\sigma'})\overline{\mathbf{s}}$ is the different contract $!(\mu x.!(x).1).1$.

Lemma 5.5. *For every $\sigma \in \mathsf{SCts}$, if σ is m-closed then $\overline{\mathsf{unfold}(\sigma)} = \mathsf{unfold}(\overline{\sigma})$.*

Proof. Let $\mathsf{unfold}(\sigma) = \rho$. We have to show that

$$\overline{\rho} = \mathsf{unfold}(\overline{\sigma}) \tag{5.1}$$

We reason by rule induction on the derivation of the judgement $\mathsf{unfold}(\sigma) = \rho$. The base case is when σ is not a recursive term, in which case ρ coincides with σ itself. Examining Definition 5.8 it

is obvious that if $\overline{\sigma}$ is a recursive term then so is σ; from this we conclude that $\mathsf{unfold}(\overline{\sigma})$ is $\overline{\sigma}$ itself, from which the required (5.1) is trivially true.

The inductive case is when σ has the form $\mu x.\sigma'$ and $\mathsf{unfold}(\sigma) = \rho$ because

$$\mathsf{unfold}(\sigma'[x \mapsto \sigma]) = \rho$$

The hypothesis that σ is m-closed, and Definition 5.3 imply that also σ' is m-closed, thus part ((c)) of Lemma 5.4 lets us prove that $\sigma'[x \mapsto \sigma]$ is m-closed. Thanks to this property of $\sigma'[x \mapsto \sigma]$ we apply the inductive hypothesis to prove that $\overline{\rho} = \mathsf{unfold}(\overline{\sigma'[x \mapsto \sigma]})$. So the required (5.1) will follow if we show $\mathsf{unfold}(\overline{\sigma}) = \mathsf{unfold}(\overline{\sigma'[x \mapsto \sigma]})$. Since σ and σ' are m-closed, part ((c)) of Lemma 5.4 ensures that

$$\overline{\sigma'[x \mapsto \sigma]} = \overline{\sigma'}[x \mapsto \overline{\sigma}] \tag{5.2}$$

The equality we are after is now easy to prove,

$$
\begin{array}{llll}
\mathsf{unfold}(\overline{\sigma}) & = & \mathsf{unfold}(\mu x.\overline{\sigma'}) & \text{Because } \overline{\sigma} = \mu x.\overline{\sigma'} \\
& = & \mathsf{unfold}(\overline{\sigma'}[x \mapsto \mu x.\overline{\sigma'}]) & \text{By definition of } \mathsf{unfold} \\
& = & \mathsf{unfold}(\overline{\sigma'}[x \mapsto \overline{\sigma}]) & \text{Because } \overline{\sigma} = \mu x.\overline{\sigma'} \\
& = & \mathsf{unfold}(\overline{\sigma'[x \mapsto \sigma]}) & \text{Because of (5.2)}
\end{array}
$$

$\square$

This last result implies the main property of $\overline{}$ over *m-closed* contracts: given a m-closed peer ρ, its dual $\overline{\rho}$ has indeed a complementary behaviour, that is $\overline{\rho}$ is in compliance with ρ, with respect to any preorder $\mathcal{B}$.

Lemma 5.6. *Suppose $\mathcal{B}$ is a reflexive relation. For every session contract ρ, we have that either* $\rho \xrightarrow{\checkmark}$ *or* $\rho \parallel \overline{\rho} \xrightarrow{\tau}_{\mathcal{B}}$.

Proof. To prove the lemma we assume that $\rho \xrightarrow{\checkmark}\!\!\!\!\!\!/\,\,$, and show that $\rho \parallel \overline{\rho} \xrightarrow{\tau}_{\mathcal{B}}$. Either $\rho \xrightarrow{\tau}$ or ρ is stable. In the first case we apply [I-Left] to derive the desired $\rho \parallel \overline{\rho} \xrightarrow{\tau}_{\mathcal{B}}$. Suppose now that ρ is stable. The argument proceeds by case analysis on the shape of ρ, which, in view of our assumption, cannot be 1 and has no top-most recursion. We discuss only three cases. If $\rho = \,!(\sigma).\rho'$, then $\overline{\rho} = \,?(\sigma).\overline{\rho'}$. Since by hypothesis $\mathcal{B}$ is reflexive, $!(\sigma) \bowtie_{\mathcal{B}} ?(\sigma)$, thus we apply [I-Synch] to derive the required $\rho \parallel \overline{\rho} \xrightarrow{\tau}_{\mathcal{B}}$. Suppose now that $\rho = \bigoplus_{i \in I} ?1_i.\rho_i$. Since ρ is stable, it must be the case that $|I| = 1$, that is $I = \{k\}$ for some k. We know by definition that $\overline{\rho} = ?1_k.\overline{\rho_k}$, and that $!1_k \bowtie_{\mathcal{B}} ?1_k$, thus we apply rule [I-Synch] to infer the hand-shake $\rho \parallel \overline{\rho} \xrightarrow{\tau}_{\mathcal{B}}$. $\square$

Theorem 5.7. *Suppose $\mathcal{B}$ is a preorder. Then $\rho \dashv\vdash^{\mathcal{B}}_{\text{p2p}} \overline{\rho}$ for every* m-closed *session contract ρ.*

Proof. Let

$$\mathcal{R} = \{ (\rho_1, \rho_2) \mid \rho \xrightarrow{\tau}{}^{\star} \rho_1, \, \overline{\rho} \xrightarrow{\tau}{}^{\star} \rho_2 \text{ for every } m\text{-closed } \rho \}$$

It is sufficient to show the set inclusion $\mathcal{R} \subseteq C^{\text{p2p}}(\mathcal{R}, \mathcal{B})$ for any preorder $\mathcal{B}$. Pick a pair of contracts (ρ_1, ρ_2) in the relation $\mathcal{R}$. The construction of $\mathcal{R}$ ensures that there exists a contract ρ which is m-closed, and such that

$$\rho \xrightarrow{\tau}{}^{\star} \rho_1, \qquad \overline{\rho} \xrightarrow{\tau}{}^{\star} \rho_2 \tag{5.3}$$

Definition 3.1 requires us to prove two properties of the pair (ρ_1, ρ_2), depending on whether the composition $\rho_1 \parallel \rho_2$ is stable or not. First assume that

$$\rho_1 \parallel \rho_2 \xrightarrow{\tau}\!\!\!\!\!\!/\,\,_{\mathcal{B}} \tag{5.4}$$

We have to show that $\rho_1 \xrightarrow{\checkmark}$ and that $\rho_2 \xrightarrow{\checkmark}$. The assumption $\rho_1 \parallel \rho_2 \xrightarrow{\tau}_{\mathcal{B}}$ ensures that $\rho_1 \xrightarrow{\tau}\!\!\!\!\!/$; a property of the unwind function, given in Lemma A.6 of the Appendix, then ensures that

$$\mathsf{unfold}(\rho) \xrightarrow{\tau}{}^{\star} \rho_1$$

Similar reasoning establishes $\mathsf{unfold}(\overline{\rho}) \xrightarrow{\tau}{}^{\star} \rho_2$. In fact by Lemma 5.2 the latter may be rewritten as $\overline{\mathsf{unfold}(\rho)} \xrightarrow{\tau}{}^{\star} \rho_2$. Introducing σ to denote $\mathsf{unfold}(\rho)$ we have now established

$$\sigma \xrightarrow{\tau}{}^{\star} \rho_1 \ \text{ and } \ \overline{\sigma} \xrightarrow{\tau}{}^{\star} \rho_2 \tag{5.5}$$

There is very little scope for performing τ transitions in (5.3) above, and even less in (5.5), since σ cannot be a recursive term of the form $\mu x. \ldots$. In fact because of the assumption (5.4) we can show that

$$\sigma = \rho_1 \ \text{ and } \ \overline{\sigma} = \rho_2 \tag{5.6}$$

The proof of this fact proceeds by a case analysis on the syntactic structure of σ. The only possibility for generating a τ transition in (5.5) is when either σ or its dual $\overline{\sigma}$ is an internal choice $\{\, !1_i.\sigma_i \mid i \in I \,\}$, where $|I| > 1$. Without loss of generality suppose the former. Then $\overline{\sigma}$ is $\sum_I ?1_i.\sigma_i$ and $\rho_1 \parallel \rho_2$ must take the form $!1_k.\sigma_k \parallel \sum_I ?1_i.\sigma_i$, for some $k \in I$. An application of [I-Synch] gives a τ transition, $\rho_1 \parallel \rho_2 \xrightarrow{\tau}_{\mathcal{B}}$, thereby contradicting the assumption (5.4).

An application of Lemma 5.6, together with the just established (5.6), immediately gives lets us show $\rho_1 \xrightarrow{\checkmark}$, from which $\rho_2 \xrightarrow{\checkmark}$ also follows, since ρ_2 is $\overline{\sigma}$.

Now suppose that $\rho_1 \parallel \rho_2 \xrightarrow{\tau}_{\mathcal{B}} \rho'_1 \parallel \rho'_2$, we have to prove that $\rho'_1 \mathrel{\mathcal{R}} \rho'_2$. This will follow if we exhibit a m-closed contract $\hat{\rho}$ such that $\hat{\rho} \xrightarrow{\tau}{}^{\star} \rho'_1$, and $\overline{\hat{\rho}} \xrightarrow{\tau}{}^{\star} \rho'_2$. We reason by case analysis on the rule of Figure 4 used to infer the silent move $\rho_1 \parallel \rho_2 \xrightarrow{\tau}_{\mathcal{B}} \rho'_1 \parallel \rho'_2$.

If [I-Left] was applied, then $\rho_1 \xrightarrow{\tau} \rho'_1$ and $\rho'_2 = \rho_2$. The desired $\hat{\rho}$ is the contract ρ itself. If rule [I-Right] again the $\hat{\rho}$ we are after is ρ itself.

The last case to discuss is when $\rho_1 \parallel \rho_2 \xrightarrow{\tau}_{\mathcal{B}} \rho'_1 \parallel \rho'_2$ is inferred applying rule [I-Synch]. In this case $\rho_1 \xrightarrow{\lambda_1} \rho'_1$, $\rho_2 \xrightarrow{\lambda_2} \rho'_2$, and $\lambda_1 \bowtie_{\mathcal{B}} \lambda_2$. We have already proven that the contract ρ_1 is m-closed, thus Part ((ii)) of Lemma 5.4 imply that ρ'_1 is m-closed. We pick as candidate $\hat{\rho}$ the contracts ρ'_1. To finish the proof it suffices to show that $\rho'_2 = \overline{\rho'_1}$. We proceed by case analysis on λ_1, and there are four cases to discuss, for λ_1 is either an input or an output, action, and there are two subcases depending on the action being first-order or higher-order. We discuss only two cases involving higher-order actions and one involving first-order actions.

Suppose that $\lambda_1 = ?(\sigma_1)$. In view of the restrictive syntax of contracts, it must be the case that $\rho_1 = ?(\sigma).\rho'_1$. Now $\rho \xrightarrow{\tau}{}^{\star} \rho'_1$ implies that $\mathsf{unfold}(\rho) = ?(\sigma).\rho'_1$.

Since $\lambda_1 \bowtie_{\mathcal{B}} \lambda_2$, $\lambda_2 = !(\sigma')$ for some σ', and $\rho_2 = ?(\sigma').\rho'_2$. The last fact and $\overline{\rho} \xrightarrow{\tau}{}^{\star} \rho_2$ imply that $\mathsf{unfold}(\overline{\rho}) = \rho_2$. An application of Lemma 5.5 lets us obtain that $\rho_2 = \mathsf{unfold}(\overline{\rho}) = !(\sigma).\overline{\rho'_1}$. It follows that $\rho'_2 = \overline{\rho'_1}$, as required.

If $\lambda_1 = !(\sigma)$, or $\lambda_1 = !\mathsf{t}$, or $\lambda_1 = ?\mathsf{t}$, the argument is analogous to the previous one.

Suppose now that $\lambda_1 = !1$. It must be the case that $\mathsf{unfold}(\rho) = \bigoplus_{i \in I} !1_i.\rho_i$, and for some $k \in I$, $\rho_1 = !1_k.\rho_k$. The construction of $\mathcal{R}$ ensures that

$$\overline{\rho} \xrightarrow{\tau}{}^{\star} \rho_2 \tag{5.7}$$

and this implies that $\mathsf{unfold}(\overline{\rho}) \xrightarrow{\tau}{}^{\star} \rho_2$. By definition $\overline{\mathsf{unfold}(\rho)} = \sum_{i \in I} ?1_i.\overline{\rho_i}$, thus commutativity (Lemma 5.5), ensures that $\mathsf{unfold}(\overline{\rho}) = \sum_{i \in I} ?1_i.\overline{\rho_i}$. Observe that $\mathsf{unfold}(\overline{\rho}) \xrightarrow{\tau}\!\!\!\!\!/$, thus Eq. (5.7) above

implies that $\rho_2 = \text{unfold}(\overline{\rho})$, so $\rho_2 = \sum_{i \in I} ?1_i.\overline{\rho_i}$. As $\rho_2 \xrightarrow{\lambda_2} \rho_2'$ and $!1_k \bowtie_{\mathcal{B}} \lambda_2$, it follows that $\lambda_2 = ?1_k$, and $\rho_2' = \overline{\rho_k}$. The equality $\rho_k = \rho_1'$ now lets us conclude that $\rho_2' = \overline{\rho_1'}$, which is the fact we were after. $\qquad\square$

5.2. **Extension to arbitrary session contracts.** Our intention here is use Theorem 5.7 to find a complement for all session contracts. This is achieved in two steps. First for each $\sigma \in \text{SCts}$ we construct a *behaviourally equivalent* contract $\text{mcl}(\sigma)$ which is *m-closed*. The required complement of σ, which we will denote by $\text{prdual}(\sigma)$ will be taken to be the standard dual of $\text{mcl}(\sigma)$.

Definition 5.8. [M-closure]
For any $\sigma \in L_{\text{SCts}}$ and any $\mathbf{s}$ such that $\text{fv}(\sigma) \subseteq \text{dom}(\mathbf{s})$ the term $\text{mclo}(\sigma, \mathbf{s})$ is defined by structural induction as follows:

$$\text{mclo}(\sigma, \mathbf{s}) = \begin{cases} 1 & \text{if } \sigma = 1, \\ x & \text{if } \sigma = x, \\ !t.\text{mclo}(\sigma', \mathbf{s}) & \text{if } \sigma = !t.\sigma', \\ ?t.\text{mclo}(\sigma', \mathbf{s}) & \text{if } \sigma = ?t.\sigma', \\ !(\sigma^m \mathbf{s}).\text{mclo}(\sigma', \mathbf{s}) & \text{if } \sigma = !(\sigma^m).\sigma', \\ ?(\sigma^m \mathbf{s}).\text{mclo}(\sigma', \mathbf{s}) & \text{if } \sigma = ?(\sigma^m).\sigma', \\ \sum_{i \in I} !1_i.\text{mclo}(\sigma_i, \mathbf{s}) & \text{if } \sigma = \sum_{i \in I} ?1_i.\sigma_i, \\ \bigoplus_{i \in I} ?1_i.\text{mclo}(\sigma_i, \mathbf{s}) & \text{if } \sigma = \bigoplus_{i \in I} !1_i.\sigma_i, \\ \mu x.\text{mclo}(\sigma', \mathbf{s} \cdot [x \mapsto \sigma\mathbf{s}]) & \text{if } \sigma = \mu x.\sigma' \end{cases}$$

Note that in the last clause the substitution $\mathbf{s} \cdot [x \mapsto \sigma\mathbf{s}]$ is well-defined, as $\sigma\mathbf{s}$ is closed.
For $\sigma \in \text{SCts}$ we let $\text{mcl}(\sigma)$, called the *m-closure* of σ, denote $\text{mclo}(\sigma, \varepsilon)$.

The intuition behind $\text{mclo}(\sigma, \mathbf{s})$ is that an m-closed term equivalent to σ, can be constructed by (1) keeping track in the accumulator $\mathbf{s}$ of all the substitutions that take place unfolding σ, and by (2) applying these substitutions only to the messages of σ, and not the continuations.

Example 5.9. In this example we apply the function $\text{mcl}(-)$ to two session contracts. The first is the contract $\rho = \mu x.?(x).1$ that we already used in Example 5.2. By definition we have the equalities

$$\begin{aligned} \text{mcl}(\rho) &= \text{mclo}(\mu x.?(x).1, \varepsilon) \\ &= \mu x.\text{mclo}(?(x).1, [x \mapsto \rho]) \\ &= \mu x.?(\rho).\text{mclo}(1, [x \mapsto \rho]) \\ &= \mu x.?(\rho).1 \end{aligned}$$

Since ρ is closed, the contract $\text{mclo}(\rho, \varepsilon)$ is m-closed.
Now we apply $\text{mcl}(-)$ to a more involved contract, namely $\sigma = \mu x.\mu y.?(y).x$. The following equalities are true by definition,

$$\begin{aligned} \text{mcl}(\sigma) &= \text{mclo}(\mu x.\mu y.?(y).x, \varepsilon) \\ &= \mu x.(\text{mclo}(\mu y.?(y).x, [x \mapsto \sigma])) \\ &= \mu x.\mu y.(\text{mclo}(?(y).x, \mathbf{s})) \\ &= \mu x.\mu y.?(\mu y.?(y).\sigma).\text{mclo}(x, \mathbf{s}) \\ &= \mu x.\mu y.?(\mu y.?(y).\sigma).x \\ &= \mu x.\mu y.?(\mu y.?(y).\sigma).x \end{aligned}$$

where $\mathbf{s} = [x \mapsto \sigma, y \mapsto \mu y.?(y).\sigma]$. Since σ is closed also the contract $\mu y.?(y).\sigma$ is closed, thus the contract $\mathsf{mclo}(\sigma, \varepsilon)$ is m-closed.

Lemma 5.10. *Suppose* $\mathsf{fv}(\sigma) \subseteq \mathsf{dom}(\mathbf{s})$. *Then* $\mathsf{mclo}(\sigma, \mathbf{s})$ *is* m-closed.

Proof. By structural induction on σ.

First suppose σ has the form $\mu x.\sigma'$. By definition $\mathsf{mclo}(\sigma, \mathbf{s}) = \mu x.\mathsf{mclo}(\sigma', \mathbf{s} \cdot [x \mapsto \sigma\mathbf{s}])$. Since $\mathsf{dom}(\mathbf{s}) \subseteq \mathsf{dom}(\mathbf{s} \cdot [x \mapsto \sigma\mathbf{s}])$, also the inclusion $\mathsf{fv}(\sigma') \subseteq \mathsf{dom}(\mathbf{s} \cdot [x \mapsto \sigma\mathbf{s}])$ is true, and we apply induction to conclude that $\mathsf{mclo}(\sigma', \mathbf{s} \cdot [x \mapsto \sigma\mathbf{s}])$ is *m-closed*; by definition this means σ is *m-closed*.

As another case suppose σ has the form $!(\sigma^m).\sigma'$. Here $\mathsf{mclo}(\sigma, \mathbf{s})$ is $!(\sigma^m\mathbf{s}).\mathsf{mclo}(\sigma', \mathbf{s})$. Induction gives that $\mathsf{mclo}(\sigma', \mathbf{s})$ is *m-closed*, and since $\mathsf{fv}(\sigma^m) \subseteq \mathsf{fv}(\mathbf{s})$ we know $\sigma^m\mathbf{s}$ is closed; this means that by definition σ is *m-closed*.

All remaining cases are either similar, or trivial. $\qquad\square$

Because of Lemma 5.10 we know from Theorem 5.7 that $\overline{\mathsf{mcl}(\rho)}$ complies with $\mathsf{mcl}(\rho)$ for every session contract ρ. We now show that that $\mathsf{mcl}(\rho)$ and ρ are behaviourally equivalent, in that they comply with exactly the same contracts. This involves first establishing a sequence of technical lemmas.

Lemma 5.11. *For every* $\sigma \in L_{\mathsf{SCts}}$, *and substitutions* $\mathbf{s}_1, \mathbf{s}_2$, *if* $\mathbf{s}_1(x) = \mathbf{s}_2(x)$ *for every* $x \in \mathsf{fv}(\sigma)$, *then* $\mathsf{mclo}(\sigma, \mathbf{s}_1) = \mathsf{mclo}(\sigma, \mathbf{s}_2)$, *whenever both are defined.*

Proof. Straightforward by structural induction on σ. $\qquad\square$

Given a substitution $\mathbf{s} = [x_1 \mapsto \sigma_1, \ldots, x_n \mapsto \sigma_n]$, we let $\mathsf{mcl}(\mathbf{s}) = [x_1 \mapsto \mathsf{mcl}(\sigma_1), \ldots, x_n \mapsto \mathsf{mcl}(\sigma_n)]$.

Proposition 5.12. *Let* $\sigma \in L_{\mathsf{SCts}}$, *and suppose that* $\mathsf{fv}(\sigma) \subseteq \mathsf{dom}(\mathbf{s}_1) \cup \mathsf{dom}(\mathbf{s}_2)$. *Then* $\mathsf{mclo}(\sigma\mathbf{s}_1, \mathbf{s}_2) = \mathsf{mclo}(\sigma, \mathbf{s}_2 \cdot \mathbf{s}_1)\mathsf{mcl}(\mathbf{s}_1)$.

Proof. By structural induction on σ. We examine the three most interesting cases.

(1) First suppose σ is a variable z. The hypothesis implies that z is either in $\mathsf{dom}(\mathbf{s}_1)$ or $\mathsf{dom}(\mathbf{s}_2)$. Suppose that $z \in \mathsf{dom}(\mathbf{s}_1)$. The left hand side is by definition $\mathsf{mclo}(\mathbf{s}_1(z), \mathbf{s}_2)$ but because $\mathbf{s}_1(z)$ is always a closed term, by Lemma 5.11, this is the same as $\mathsf{mclo}(\mathbf{s}_1(z), \varepsilon)$, that is $\mathsf{mcl}(\mathbf{s}_1(z))$. This is precisely the right hand side: by definition $\mathsf{mclo}(z, \mathbf{s}_2 \cdot \mathbf{s}_1) = z$, so applying the substitution $\mathsf{mcl}(\mathbf{s}_1)$ to z we have $\mathsf{mcl}(\mathbf{s}_1(z))$.

On the other hand if $z \in \mathsf{dom}(\mathbf{s}_2)$ and $z \notin \mathsf{dom}(\mathbf{s}_1)$ then both sides evaluate to the term z.

(2) Suppose σ has the form $\mu x.\sigma'$. Here $\mathsf{mclo}(\sigma, \mathbf{s}_2 \cdot \mathbf{s}_1) = \mu x.\mathsf{mclo}(\sigma', \mathbf{s}_2 \cdot \mathbf{s}_1 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)])$. So the right hand side is equal to

$$(\mu x.(\mathsf{mclo}(\sigma', \mathbf{s}_2 \cdot \mathbf{s}_1 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)])))\,\mathsf{mcl}(\mathbf{s}_1)$$

Applying the definition of substitution we get

$$\mu x.(\mathsf{mclo}(\sigma', \mathbf{s}_2 \cdot \mathbf{s}_1 \cdot [x \mapsto \sigma\mathbf{s}_2 \cdot \mathbf{s}_1])\mathsf{mcl}(\mathbf{s}_1^{\backslash x})) \tag{5.8}$$

We show that the left hand side can be rewritten in this form also. First we apply the substitution $\mathbf{s}_1$ to σ and get

$$\sigma\,\mathbf{s}_1 = \mu x.(\sigma'\,\mathbf{s}_1^{\backslash x})$$

Then applying the definition of $\mathsf{mclo}(-, -)$ we have

$$\mathsf{mclo}(\sigma\mathbf{s}_1, \mathbf{s}_2) = \mu x.(\,\mathsf{mclo}(\sigma'\mathbf{s}_1^{\backslash x}, \mathbf{s}_2 \cdot [x \mapsto (\sigma\mathbf{s}_1)\mathbf{s}_2])\,)$$

$$= \mu x.(\,\mathsf{mclo}(\sigma'\mathbf{s}_1^{\backslash x}, \mathbf{s}_2 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)])\,) \tag{5.9}$$

Since $\mathsf{fv}(\sigma') \subseteq \mathsf{dom}(\mathbf{s}_1^{\backslash x}) \cup \mathsf{dom}(\mathbf{s}_2 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)])$ induction lets us infer

$$\mathsf{mclo}(\sigma'\mathbf{s}_1^{\backslash x}, \mathbf{s}_2 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)]) = \mathsf{mclo}(\sigma', \mathbf{s}_2 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)] \cdot \mathbf{s}_1^{\backslash x}) \, \mathsf{mcl}(\mathbf{s}_1^{\backslash x})$$

The substitution $[x \mapsto \rho] \cdot \mathbf{s}_1^{\backslash x}$ can also be written as $\mathbf{s}_1 \cdot [x \mapsto \rho]$, for any ρ, thus

$$\mathbf{s}_2 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)] \cdot \mathbf{s}_1^{\backslash x} = \mathbf{s}_2 \cdot \mathbf{s}_1 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)]$$

which means that rewriting (5.9) above, we get

$$\mathsf{mclo}(\sigma\mathbf{s}_1, \mathbf{s}_2) = \mu x.(\mathsf{mclo}(\sigma', \mathbf{s}_2 \cdot \mathbf{s}_1 \cdot [x \mapsto \sigma(\mathbf{s}_2 \cdot \mathbf{s}_1)])\mathsf{mcl}(\mathbf{s}_1^{\backslash x}))$$

Thus the left hand side coincides with (5.8) above, as required.

(3) Suppose σ has the form $?(\sigma^m).\sigma'$. By definition $\mathsf{mclo}(\sigma, \mathbf{s}_2 \cdot \mathbf{s}_1) = ?(\sigma^m (\mathbf{s}_2 \cdot \mathbf{s}_1)).\mathsf{mclo}(\sigma', \mathbf{s}_2 \cdot \mathbf{s}_1)$ and we know by Lemma 5.10 that $\sigma^m (\mathbf{s}_2 \cdot \mathbf{s}_1)$ is a closed term. So the right hand side can be written as

$$?(\sigma^m (\mathbf{s}_2 \cdot \mathbf{s}_1)).(\mathsf{mclo}(\sigma', \mathbf{s}_2 \cdot \mathbf{s}_1) \, \mathsf{mcl}(\mathbf{s}_1))$$

Here induction can be applied to the residual, and therefore the right hand side now looks like

$$?(\sigma^m (\mathbf{s}_2 \cdot \mathbf{s}_1)). \, \mathsf{mclo}(\sigma'\mathbf{s}_1, \mathbf{s}_2)$$

But this is exactly $\mathsf{mclo}(\sigma\mathbf{s}_1, \mathbf{s}_2)$, that is the left hand side. $\qquad \square$

We will use a specific instance of this Proposition, captured in the following Corollary:

Corollary 5.13. *For every $\sigma_2 \in \mathsf{SCts}$, and $\sigma_1 \in L_{\mathsf{SCts}}$, if $\mathsf{fv}(\sigma_1) \subseteq \{x\}$, then*

$$\mathsf{mcl}(\sigma_1[x \mapsto \sigma_2]) = \mathsf{mclo}(\sigma_1, [x \mapsto \sigma_2]) [x \mapsto \mathsf{mcl}(\sigma_2)]$$

Proof. An immediate application of Proposition 5.12, with $\mathbf{s}_2$ instantiated to the empty substitution, and $\mathbf{s}_1$ the singleton substitution $[x \mapsto \sigma_2]$. $\qquad \square$

Proposition 5.14. *Let $\sigma_1 \in \mathsf{SCts}$.*

(1) $\sigma_1 \xrightarrow{\checkmark}$ *if and only if* $\mathsf{mcl}(\sigma_1) \xrightarrow{\checkmark}$, *moreover*
(2) *for every $\mu \in \mathsf{Act}_\tau$,*

   (a) $\sigma_1 \xrightarrow{\mu} \sigma_2$ *implies* $\mathsf{mcl}(\sigma_1) \xrightarrow{\mu} \mathsf{mcl}(\sigma_2)$
   (b) *conversely,* $\mathsf{mcl}(\sigma_1) \xrightarrow{\mu} \sigma'$ *implies* $\sigma' = \mathsf{mcl}(\sigma_2)$ *for some σ_2 satisfying* $\sigma_1 \xrightarrow{\mu} \sigma_2$

Proof. Suppose that $\sigma_1 \xrightarrow{\checkmark}$, the syntax of session contracts implies that $\sigma_1 = 1$, thus by definition $\mathsf{mcl}(\sigma_1) = 1$, hence $\mathsf{mcl}(\sigma_1) \xrightarrow{\checkmark}$. Conversely, if $\mathsf{mcl}(\sigma_1) \xrightarrow{\checkmark}$ then the syntax ensures that $\mathsf{mcl}(\sigma_1) = 1$, thus the definition of $\mathsf{mcl}(-)$ implies that $\sigma_1 = 1$, and plainly $\sigma_1 \xrightarrow{\checkmark}$.

Now let $\mu \in \mathsf{Act}_\tau$. The proofs of both (1) and (2) are by structural induction on σ_1. We look briefly at (2). Observe that $\mathsf{mcl}(\sigma_1) \xrightarrow{\mu}$ ensures that $\sigma_1 \neq 1$.

(a) Suppose $\sigma_1 = !(\sigma^m).\sigma'_1$. By definition $\mathsf{mcl}(\sigma_1) = !(\sigma^m).\mathsf{mcl}(\sigma'_1)$, hence the contract $\mathsf{mcl}(\sigma_1)$ performs only the action $\mathsf{mcl}(\sigma_1) \xrightarrow{!(\sigma^m)} \sigma'$ with $\sigma' = \mathsf{mcl}(\sigma'_1)$. By letting $\sigma_2 = \sigma'_1$ we obtain immediately $\sigma' = \mathsf{mcl}(\sigma_2)$, and $\sigma_1 \xrightarrow{!(\sigma^m)} \sigma_2$.

(b) Suppose σ_1 is $\mu x.\sigma'_1$. Here $\mathsf{mcl}(\sigma_1)$ by definition is $\mu x.\mathsf{mclo}(\sigma'_1, [x \mapsto \sigma_1])$. So the only possible move $\mathsf{mcl}(\sigma_1) \xrightarrow{\mu} \sigma'$ is with $\mu = \tau$ and σ' of the form $\mathsf{mclo}(\sigma'_1, [x \mapsto \sigma_1]) [x \mapsto \mathsf{mcl}(\sigma_1)]$. By Corollary 5.13 this is the same as $\mathsf{mcl}(\sigma'_1[x \mapsto \sigma_1])$. Setting σ_2 to be $\sigma'_1 [x \mapsto \sigma_1]$ the result follows, because $\sigma_1 \xrightarrow{\tau} (\sigma'_1 [x \mapsto \sigma_1])$.

(c) All other possibilities for σ are treated as in case ((iv)).

$\square$

Proposition 5.15. *For every session contract* $\sigma \in \mathsf{SCts}$, $\sigma \sim \mathsf{mcl}(\sigma)$.

Proof. Let $\mathcal{R}$ be the set $\{\,(\sigma, \mathsf{mcl}(\sigma)) \mid \sigma \in \mathsf{SCts}\,\}$. It is straightforward to use Proposition 5.14 to show that $\mathcal{R}$ is a strong bisimulation, as given in Definition 3.3. $\square$

Definition 5.16. [Peer-duality] For every session contract $\rho \in \mathsf{SCts}$, let $\mathsf{prdual}(\rho)$ denote $\overline{\mathsf{mcl}(\rho)}$.

Theorem 5.17. *Suppose* $\mathcal{B}$ *is a preorder. Then* $\rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \mathsf{prdual}(\rho)$ *for every session contract* $\rho \in \mathsf{SCts}$.

Proof. From Lemma 5.10 we know that $\mathsf{mcl}(\rho)$ is *m-closed* and therefore an application of Theorem 5.7 gives $\mathsf{mcl}(\rho) \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \mathsf{prdual}(\rho)$.

We also know from Proposition 5.15 that $\rho \sim \mathsf{mcl}(\rho)$, and so the required $\rho \dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}} \mathsf{prdual}(\rho)$ follows by Proposition 3.5. $\square$

5.3. **The complement function.** In the original extended abstract of the current paper, [BH14] we proposed an alternative function $\mathsf{cplmt}(-)$ in order to construct the complementary contracts required by Theorem 3.16; this function was proposed independently in [BP12], but for a different purpose.

In this section we show that the function $\mathsf{cplmt}(-)$ suffers the same issue of the standard duality: there exists a σ which is not in $\mathcal{B}$-peer compliance with its proposed complement $\mathsf{cplmt}(\sigma)$, for every reasonable $\mathcal{B}$ (Example 5.21).

We begin by recalling the necessary definitions, and then we present a series of examples.

Definition 5.18. [Complement]
Let $\mathsf{cplmt} : L_{\mathsf{SCts}} \longrightarrow L_{\mathsf{SCts}}$ be defined inductively as follows,

$$\mathsf{cplmt}(\sigma) = \begin{cases} 1 & \text{if } \sigma = 1, \\ x & \text{if } \sigma = x, \\ ?t.\mathsf{cplmt}(\sigma') & \text{if } \sigma = !t.\sigma', \\ !t.\mathsf{cplmt}(\sigma') & \text{if } \sigma = ?t.\sigma', \\ ?(\sigma^m).\mathsf{cplmt}(\sigma') & \text{if } \sigma = !(\sigma^m).\sigma', \\ !(\sigma^m).\mathsf{cplmt}(\sigma') & \text{if } \sigma = ?(\sigma^m).\sigma', \\ \bigoplus_{i \in I} !1_i.\mathsf{cplmt}(\sigma_i) & \text{if } \sigma = \sum_{i \in I} ?1_i.\sigma_i, \\ \sum_{i \in I} ?1_i.\mathsf{cplmt}(\sigma_i) & \text{if } \sigma = \bigoplus_{i \in I} !1_i.\sigma_i, \\ \mu x.\mathsf{cplmt}(\sigma' \lfloor \sigma \mapsto x \rfloor) & \text{if } \sigma = \mu x.\sigma' \end{cases}$$

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

In the definition above the application of $\lfloor \sigma \mapsto x \rfloor$ to σ' stands for the substitution of σ in place of x in the message fields that appear in σ'. The definition is the following,

$$\rho\lfloor\sigma\mapsto x\rfloor = \begin{cases} 1 & \text{if } \rho = 1, \\ y & \text{if } \rho = y, \\ ?t.(\rho'\lfloor\sigma\mapsto x\rfloor) & \text{if } \rho = ?t.\rho', \\ !t.(\rho'\lfloor\sigma\mapsto x\rfloor) & \text{if } \rho = !t.\rho', \\ ?(\rho''\lfloor\sigma\mapsto x\rfloor).(\rho'\lfloor\sigma\mapsto x\rfloor) & \text{if } \rho = ?(\rho'').\rho', \\ !(\rho''\lfloor\sigma\mapsto x\rfloor).(\rho'\lfloor\sigma\mapsto x\rfloor) & \text{if } \rho = !(\rho'').\rho', \\ \mu y.(\rho'\lfloor\sigma\mapsto x\rfloor) & \text{if } \rho = \mu y.\rho' \text{ and } y \neq x, \\ \mu x.\rho' & \text{if } \rho = \mu x.\rho' \end{cases}$$

Example 5.19 (Inner substitution)**.** In this example we show how the application of inner substitutions acts on terms. Fix a term $\sigma \in \mathsf{SCts}$, by definition $(?(y).y)\lfloor\sigma\mapsto y\rfloor = ?(\sigma).y$. This equality shows that inner substitutions operate on the variables that appear free in the message parts of terms, and not on the ones in the continuations. We have likewise the equality $(?(y).x)\lfloor\sigma\mapsto x\rfloor = ?(y).x$ because x appears free only in the continuation of the term $?(y).x$. As usual the substitution does not act on closed variables, thus $(\mu y.?(y).x)\lfloor\sigma\mapsto y\rfloor = \mu y.?(y).x$.

As the application of $\lfloor\sigma\mapsto x\rfloor$ does not change the number of prefixes that appear in a term, $\mathsf{cplmt}(\sigma)$ is defined for every session type term σ.

Example 5.20 (Complement function)**.** In this example we prove that in general the function $\mathsf{cplmt}(-)$ does not compute m-closed terms. Recall the type σ we used in Example 5.9, namely $\sigma = \mu x.\sigma'$, where $\sigma' = \mu y.?(y).x$. Calculations let us prove the equality

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

The crucial observation here is that the contract $\mathsf{cplmt}(\sigma)$ is not m-closed, for it contains the message $\mu y.?(y).x$, which is an open term.

Example 5.21. In this example we show a contract σ such that $\sigma \not\vdash^{\mathcal{B}}_{\mathrm{P2P}} \mathsf{cplmt}(\sigma)$ for every reasonable $\mathcal{B}$. Recall the type $\sigma = \mu x.\mu y.?(y).x$ of Example 5.9. In Example 5.20 we have proven that

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

Fix a reasonable relation $\mathcal{B}$. We want to prove that $\sigma \not\vdash^{\mathcal{B}}_{\mathrm{P2P}} \mathsf{cplmt}(\sigma)$. Since $\sigma \xrightarrow{\tau} \mathsf{unfold}(\sigma)$ and $\mathsf{cplmt}(\sigma) \xrightarrow{\tau} \mathsf{unfold}(\mathsf{cplmt}(\sigma))$, it suffices to prove that

$$\mathsf{unfold}(\sigma) \not\vdash^{\mathcal{B}}_{\mathrm{P2P}} \mathsf{unfold}(\mathsf{cplmt}(\sigma)) \tag{5.10}$$

Let us unfold the terms σ and $\mathsf{cplmt}(\sigma)$,

$$\begin{aligned} \mathsf{unfold}(\sigma) \quad &= \quad \mathsf{unfold}(\mu x.\mu y.?(y).x) \\ &= \quad \mathsf{unfold}(\mu y.?(y).\sigma) \\ &= \quad ?(\mu y.?(y).\sigma).\sigma \end{aligned}$$

$$\begin{aligned} \mathsf{unfold}(\mathsf{cplmt}(\sigma)) \quad &= \quad \mathsf{unfold}(\mu x.\mu y.!(\mu y.?(y).x).x) \\ &= \quad \mathsf{unfold}(\mu y.!(\mu y.?(y).\mathsf{cplmt}(\sigma)).\mathsf{cplmt}(\sigma)) \\ &= \quad !(\mu y.?(y).\mathsf{cplmt}(\sigma)).\mathsf{cplmt}(\sigma) \end{aligned}$$

As preliminary fact, observe that $(\mathsf{unfold}(\mathsf{cplmt}(\sigma)), \mathsf{unfold}(\sigma)) \notin \mathcal{B}$, because $\mathsf{unfold}(\mathsf{cplmt}(\sigma))$ performs an output action, $\mathsf{unfold}(\sigma)$ performs an input action, and $\mathcal{B}$ is reasonable. It follows that $(\mathsf{cplmt}(\sigma), \sigma) \notin \mathcal{B}$.

$$\frac{\Theta\,;\,\Gamma \vdash P \rhd \Delta \cdot \kappa^+ : T^+ \cdot \kappa^- : T^-}{\Theta\,;\,\Gamma \vdash (\nu\kappa)\,P \rhd \Delta} \quad T^+ \; \mathcal{D} \; T^- \quad [\text{CRes}]$$

Figure 7: The use of duality in type inference á la [YV07]

Let $\sigma_1 = \mu y.?(y).\mathsf{cplmt}(\sigma)$ and $\sigma_2 = \mu y.?(y).\sigma$. Plainly, $\mathsf{unfold}(\mathsf{cplmt}(\sigma)) = !(\sigma_1).\mathsf{cplmt}(\sigma)$, and $\mathsf{unfold}(\sigma) = ?(\sigma_2).\sigma$.

Since $\mathsf{unfold}(\sigma) \overset{\checkmark}{\nrightarrow}$, Eq. (5.10) will follow if we show that

$$\mathsf{unfold}(\sigma) \parallel \mathsf{unfold}(\mathsf{cplmt}(\sigma)) \overset{\tau}{\nrightarrow}_{\mathcal{B}}$$

Thanks to the syntax of the contracts $\mathsf{unfold}(\sigma), \mathsf{unfold}(\mathsf{cplmt}(\sigma))$, and rule [I-Synch], we have to show that $!(\sigma_1) \not\bowtie_{\mathcal{B}} ?(\sigma_2)$. In view of the definition of $\bowtie_{\mathcal{B}}$ we have to prove that $(\sigma_1, \sigma_2) \notin \mathcal{B}$. The visible moves $\mathsf{unfold}(\sigma_1) \overset{?(\sigma_1)}{\longrightarrow} \mathsf{cplmt}(\sigma)$, and $\mathsf{unfold}(\sigma_2) \overset{?(\sigma_2)}{\longrightarrow} \sigma$, together with $(\mathsf{cplmt}(\sigma), \sigma) \notin \mathcal{B}$, and part (iii)) of Definition 5.1, ensure the desired $(\sigma_1, \sigma_2) \notin \mathcal{B}$, thus $\sigma \not\vdash_{\text{P2P}}^{\mathcal{B}} \mathsf{cplmt}(\sigma)$.

The argument used in the previous example can be adapted to show that that the complement function on session types does not commute with the unfolding function:

there exists a $T \in \mathsf{Typ}$, such that $\mathsf{cplmt}(\mathsf{unfold}(T)) \neq_{\text{eq}} \mathsf{unfold}(\mathsf{cplmt}(T))$

5.4. **Discussion.** Here we briefly discuss the use of the duality operator $\overline{T}$ in type-checking systems for session types. For processes we use the syntax of [YV07], and the type-checking rules given on page 89, and in Figure 6 on page 80 of the same paper. For convenience we display in Figure 7 a slightly generalised version of the main rule involving duality; there κ^+, κ^- must have associated types T^+, T^- satisfying $T^+ \; \mathcal{D} \; T^-$. Here $\mathcal{D}$ is some relation over types which intuitively captures the notion of ensuring complementary behaviour. In [YV07] this is actually instantiated to duality: $T^+ \; \mathcal{D} \; T^-$ if $\overline{T^+} = T^-$.

Let us now reconsider the program P in Example 1.2 from the Introduction and discuss, informally, how it can be assigned a type. In order to use this instantiation of the rule in Figure 7 we need to assign to $\kappa_{\mathbf{f}}^+, \kappa_{\mathbf{f}}^-$ types satisfying $T^+ \; \mathcal{D} \; T^-$. For convenience we work up to the type equivalence $=_{\text{eq}}$ generated by the subtyping relation.

Assume that z be at type T_z (this could be stated in the syntax itself by using the annotation $z : T_z$). Since z replaces the formal parameter x in the recursion $X[\,z, \kappa_{\mathbf{f}}^-\,]$, one expects T_x, the type of x, to be equivalent to T_z, $T_x =_{\text{eq}} T_z$. By inspecting the syntax

$$\texttt{throw } x[\,\kappa_{\mathbf{f}}^+\,];\mathbf{0} \parallel \texttt{catch } y(z) \texttt{ in } \ldots$$

we also know that the endpoint x is used according to the type $T_x =_{\text{eq}} !(T^+).\text{END}$, and that T^+ must be a subtype of T_z, and so of T_x, $T^+ \leqslant T_x$. The last inequality is trivially satisfied by letting $T^+ =_{\text{eq}} T_x$, and this leaves us with the equation $T^+ =_{\text{eq}} !(T^+).\text{END}$. A session type that satisfies it is the following one,

$$T^+ = \mu X.!(X).\text{END}$$

For P to be typable it is necessary also that the type of $\kappa_{\mathbf{f}}^-$, namely T^-, be complementary to T^+. Since in $X[\,z, \kappa_{\mathbf{f}}^-\,]$ the endpoint $\kappa_{\mathbf{f}}^-$ replaces y, it must be the case that $T^- =_{\text{eq}} T_y$. At each iteration y is used to read *only once* an endpoint of type T_z, so

$$T_y =_{\text{eq}} ?(T_z).\text{END} =_{\text{eq}} ?(T_x).\text{END} =_{\text{eq}} ?(T^+).\text{END}$$

and thus $T^- = ?(T^+).\textsc{end}$. It may seem counter-intuitive that the behaviour of a process that uses an endpoint according to the type T^+, i.e. $\mu X.!(X).\textsc{end}$, be complementary to the behaviour of a process that uses the other endpoint according to type T^-, i.e. $?(T^+).\textsc{end}$. But this is easily understood if we unfold the types:

$$\text{unfold}(T^+) \;=\; !(T^+).\textsc{end}$$
$$\text{unfold}(T^-) \;=\; ?(T^+).\textsc{end}$$

More formally, since $\mathcal{M}(T^-) = \text{unfold}(\mathcal{M}(\text{prdual}(T^+)))$ Theorem 5.17 ensures that for any preorder $\mathcal{B}$, we have $\mathcal{M}(T^-) \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \mathcal{M}(T^+)$, and by using the types T^+ and T^- we can type P (see Appendix B). Our discussion shows that in general if an endpoint κ^+ is used as prescribed by a type T^+, then the other endpoint of the session, i.e. κ^+, needs to be used according to a type T^- which contains the type of κ^+. In other terms, to know how a system uses κ^- one needs to know how the system uses κ^+.

Our proposal is to amend the type-checking system in [YV07] by using the variation of the rule [CRes] in Figure 7 where

$$T^+ \;\mathcal{D}\; T^- \quad \text{whenever } \text{prdual}(T^+) = T^- \tag{5.11}$$

Here we use prdual in the obvious manner as an operator on types although formally it has only been defined on contracts. This version of the rule has a behavioural justification due to Theorem 5.17 and the interpretation $\mathcal{M}$ of types as contracts. It ensures that T^+ interpreted as a contract is in $\mathcal{B}$-mutual compliance with T^-: $\mathcal{M}(T^+) \dashv\vdash^{\mathcal{B}}_{\text{P2P}} \mathcal{M}(T^-)$.

This version of the rule, using prdual also leads to a more powerful type-checking system. Using it we can type the program P from Example 1.2. In Appendix B we give all the details of a derivation tree for of $\vdash P$, but using the generic rule from Figure 7. The construction of this derivation tree gives rise to a series of conditions on types, which boil down to:

$$T_x =_{\text{eq}} !(T_x).\textsc{end} \tag{5.12}$$

$$T_y =_{\text{eq}} ?(T_x).\textsc{end} \tag{5.13}$$

$$T_x \;\mathcal{D}\; T_y \tag{5.14}$$

We have already argued that (5.12) is satisfied by T^+, and that (5.13) is satisfied by the T^- we discussed earlier on, so if we choose $\mathcal{D}$ as in (5.11) above (5.14) is also satisfied. Thus the derivation of $\vdash P$ exists with our suggested modified inference rule.

If we choose $\mathcal{D}$ to be the standard duality, then (5.14) is not satisfied, because $\overline{T^+} \neq_{\text{eq}} T^-$, and so we cannot derive $\vdash P$.

Thus far, we have discussed only session types. The type systems that use session types, though, use the duality also to type channel (i.e. non session) types.

As an example, consider the type discipline of [Vas12]. In that paper channels are resources that can be replicated, while session endpoints are resources that cannot be replicated. This distinction is borne out by the types, which are pairs (q, T), or simply $q\,T$, where q is a qualifier that can be either un (unbound) or lin (linear), and p is a *pretype*. Pretypes are elements of STyp.

Example 5.22. [Replication and ever-lasting communications]
we write the next process using the syntax of [Vas12],

$$Q = (\nu xy)\,(\,\overline{x}\langle\, x\,\rangle.\mathbf{0} \parallel \text{un } y(z).\overline{z}\langle\, z\,\rangle.\mathbf{0}\,)$$

The process Q creates two fresh names x and y, then sends x sends over itself, to the process un $y(z).\overline{z}\langle\, z\,\rangle.\mathbf{0}$, which is a replicated input. Upon reception of x over y, the replicated input reduces to $\overline{x}\langle\, x\,\rangle.\mathbf{0} \parallel \text{un } y(z).\overline{z}\langle\, z\,\rangle.\mathbf{0}$. This entails a livelock, and no communication takes place.

$$\frac{\Gamma, x\colon T^+, y\colon T^- \vdash P}{\Gamma \vdash (\nu xy)P} \quad T^+ \, \mathcal{D} \, T^- \quad \text{[T-Res]}$$

Figure 8: The use of duality in type inference á la [Vas12]

Since x sends itself, it must be an unbound resource, that is it can be duplicated. The syntax $\mathsf{un}\, y(\,z\,).\bar{z}\langle\,z\,\rangle.\mathbf{0}$ means that also the name y can be duplicated. In turn the types of both x and y will be decorated with the qualifier un.

To show that $\vdash Q$ we use the derivation rules of [Vas12], but replacing the rule that depends on the duality, namely [T-Res] of that paper, with the rule in Figure 8. The derivation of $\vdash Q$, whose details are in Appendix B, depends on the satisfaction of three requirements, namely

$$T_x =_{\mathsf{eq}} \mathsf{un}\, !(T_x).\text{END}, \quad T_y =_{\mathsf{eq}} \mathsf{un}\, ?(T_x).\text{END}, \quad T_x \, \mathcal{D} \, T_y$$

Also in this case the power of the type system can be improved by using prdual in place of the standard duality. If, modulo the qualifiers, we instantiate $\mathcal{D}$ as in (5.11) then we can derive $\vdash Q$, while if we instantiate $\mathcal{D}$ to the standard type duality, then we cannot derive $\vdash Q$. The details are in Appendix B.

6. Conclusion

In this paper we proposed a new model for recursive higher-order session types [HVK98], which is fully-abstract with respect to the subtyping relation [GH05]. The interpretation of session types maps them into higher-order session contracts. This is a sublanguage of higher-order contracts for web-services.

To construct the model, we have equipped those contracts with a novel behavioural theory. In our theory the observable behaviour of contracts is expressed via a standard LTS, but the interactions of contracts are parametrised over preorders $\mathcal{B}$s. The result is a family of LTS. For each one of them we defined a mutual compliance, $\dashv\vdash^{\mathcal{B}}_{\mathrm{P2P}}$. Then we defined a family of behavioural preorders, $\sqsubseteq^{\mathcal{B}}$, such that $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ is all the contracts in $\mathcal{B}$-mutual compliance with σ_1 is in $\mathcal{B}$-mutual compliance with σ_2. The preorder that models the subtyping is the greatest solution of the equation $X = \sqsubseteq^X$, and Theorem 4.10 shows that the model is fully-abstract.

The technical development relies on a coinductive syntactic characterisation of the preorders $\sqsubseteq^{\mathcal{B}}$ (Corollary 3.19). The completeness of the characterisation depends on the existence for every contract ρ of a contract σ in B-mutual compliance with ρ, at least when $\mathcal{B}$ is a preorder. To prove this we introduced a novel function called *peer-dual*, and we have shown that this function improves on the standard duality, in that it allows us to type more well-formed processes.

Moreover, the examples in Section 5.4 suggest that type checking algorithms based on the standard inductive duality, are not complete with respect to type disciplines based on the *coinductive duality* of [GH05, Def. 9].

In summary, the contributions of this paper are two, namely

- the first fully-abstract model of the subtyping for session types of [GH05],
- the definition of a novel type duality, which leads to type systems more powerful than the one relying on the standard definition of type duality [HVK98].

6.1. **Related work.** The material in sections 2,3, and 4 is adapted from chapter 8 of [Ber13]. Lemma 4.9 in this paper is analogous to Lemma 8.1.5 of [Ber13], of that thesis. Lemma 8.1.5 relies on Lemma 7.2.18 (also of [Ber13]) which turns out to be false: and a counter example is the session contract $\mu x.!(x).1$. Spurred on by this problem, we investigated a novel definition of type duality, that we described in Section 5.

Contracts for web-services: First-order contracts for web-services and the notion of contract compliance have been proposed first in [CCLP06], and have been improved on in [LP07]. In [CCLP06] a sub-contract relation is defined, which leads to the definition of compliance. In contracts, in [LP07] the compliance is defined in terms of the LTS of contracts, and then, in the style of testing theory [DH84], 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 [Pad10, CGP09]. A striking difference between the two papers is the treatment of infinite behaviours. In [Pad10] infinite behaviours are expressed by recursive contracts, whereas in [CGP09] 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 [Pad10].

Both [Pad10, CGP09] define a subcontract preorder for contracts of server processes, a more generous *weak* subcontract. They propose mechanisms to coerce contracts, namely orchestrators [Pad10] and filters [CGP09], and show that if two contracts, σ_1, σ_2 are in the weak subcontract, then there exists a coercion f such that σ_1 and $f(\sigma_2)$ are in the subcontract relation [Pad10, Corollary 3.11], [CGP09, Theorem 3.9].

The authors of [CGP09] introduce also a compliance for processes, and show that if contracts are associated to processes by a relation that satisfies a number of properties (Definition 4.2 in that paper), then two processes are in compliance if the associated contracts are in compliance (see Theorem 4.5 there).

Session types: Session types has been presented for the first time in [THK94]. There the language for types is higher-order and without recursion, and the type duality is defined inductively in the obvious way. The main result of [THK94] is that well-typed programs cannot incur in communication errors (see Theorem 5.10 there). This type-safety result is a landmark of session types, and is proven is almost every presentation of session typed languages.

The original presentation of [THK94] has been extended extended to recursive higher-order session types in [HVK98], where also the definition of type duality that we reported in Figure 5 has been proposed. The authors of [HVK98] argue in favour of program abstractions, that help programmers structure the interaction of processes around sessions. As in [THK94], the proposed result is that a "typable program never reduces into an error" (see Theorem 5.4 (3) of [HVK98]). In [YV07, pag. 86, paragraph 4], though, it is shown that that result is not true, that is the type system of [HVK98] does not satisfy type-safety. The authors of [YV07] amend the type system of [HVK98], thereby achieving type-safety (see Theorem 3.4 of [YV07]).

Subtyping for recursive higher-order session types has been introduced in [GH05], 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 soundness, 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 $\mu X.!(X).\text{END}$. The consequence is that there is the possibility that the algorithm of

```
type T = !T.end

def f (x: T, y: dualof T) =
        new a b: T
        x!a |
        y?z. f!(z, b)

{}
```

Figure 9: Script in SEPI for the process *P* of Example 1.2

[GH05], if employed in more general settings, may not be sound, that is reject programs which are well-typed.

An alternative "fair" subtyping has been proposed recently in [Pad11]. 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 of [Pad11]. However, our aim was to model the standard subtyping of [GH05], while Padovani focuses on the properties of his fair subtyping.

An overview of the major development of session types is [DCd09].

Session types have been fertile ground for theoretical studies as well as for implementations. For instance, a programming language equipped with session types is SEPI [FV13]. In SEPI, the process *P* of Example 1.2 is rendered by the code in Figure 9, and the type checker accepts *P*, because it uses the *coinductive* duality rather than the inductive one [Vas13].

Models of Gay & Hole subtyping: The first attempt to model the Gay & Hole subtyping of [GH05] in terms of a compliance preorder appeared in [LP08]. For a comparison of that research and our work the reader is referred to [BH12]. The authors of [Bd10] 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 LTSs the orthogonality of [Bd10].
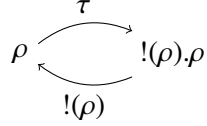
Following the approach of [Bd10], in [BH12] we have shown a fully-abstract model of the subtyping, but using the standard asymmetric compliance and an intersection of the obvious server and client preorders.

An alternative definition of the model proposed in [BH12] can be found in [Ber13, Chapter 5], where must testing of [DH84] is used in place of the compliance. This entails immediately the differences between the standard subtyping of [GH05] and the fair one of [Pad11], for they are just the differences between must preorder and the should preorder.

This paper subsumes [BH12] in the sense that Theorem 4.10 of this work implies Theorem 5.2 of [BH12].

The authors of [Bd10] have also extended their work to higher-order session types. However in their recent work [BdL13] only a subset of types is modelled, for instance the types $\mu X.!(X).\text{END}$ is ruled out, and has no behavioural interpretation at all. This is because the authors of [BdL13] use the standard definition of syntactic duality, and define the LTS of contracts by stratification. In contrast with [BdL13], we have argued that types as $\mu X.!(X).\text{END}$ are necessary to type a series of

well-formed processes. Because of this, we have replaced the standard type duality function, $\overline{\sigma}$, with our novel *peer-duality* function, $\mathsf{prdual}(\sigma)$, and we have defined the LTS of contracts *coinductively*. As a result, the type $\mu X.!(X).\textsc{end}$ in our theory is given the following observable behaviour, where $\rho = \mathcal{M}(\mu X.!(X).\textsc{end})$:

$$\rho \underset{!(\rho)}{\overset{\tau}{\rightleftarrows}} \ !(\rho).\rho$$

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 [FCB08] 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 [CDV08], 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 [CDCGP09]; 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 $?(\mathtt{Int}).\textsc{end} \leqslant ?(\mathtt{Real}).\textsc{end}$ whereas $?(\mathtt{Int}).\textsc{end} \not\leq ?(\mathtt{Real}).\textsc{end}$. The particular use of *stratification* (Theorem 2.6) is also complex, and rules out the use of session types such as $\mu X.!(X).\textsc{end}$. Finally they use as types infinite regular trees whereas we prefer to work directly with recursive terms, as proposed in [GH05]; for example this allows us to discuss the inadequacies of the type-checking rules of [YV07].

Nevertheless the extended language of sessions types of [CDCGP09] 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 [GH05].

Further behavioural models: Recently, a behavioural model for multi-party first-order session types appeared [DY13], which is based on communicating automata rather than contracts for web-services. The focus of [DY13] is not to model the subtyping.

6.2. **Future work.** In [BH12], building on [Bd10], we have already developed a result similar to the full-abstraction of Section 4, but for for *first-order* session contracts, and using a combination of *server* and *client* subcontract relations. We leave as future work showing how this approach can be recovered from our *peer* subcontract relation.

Even though standard models of recursive types are based on regular trees [PS96, BH98], tree models for recursive higher-order session types are still lacking. We plan to develop such a model, and to show the connection with the notions we used in this paper. Establishing such a connection will help us motivating the complement function, and showing its connection with other notions of duality, for instance the co-inductive one of [GH05, Def. 9].

Recently, type disciplines based on session types have been proposed, which guarantee that well-typed programs are free from deadlocks [DCdY07, Wad12, VV13]. However, all those papers deal but with finite (i.e. non recursive) types. We plan to investigate whether the semantic techniques we used in this work lead to type systems for recursive session types that guarantee deadlock freedom.

Acknowledgements. The authors would like to thank Vasco Vasconcelos for having provided the SePI code shown in Figure (9), for his remarks on the SePI type checker, and for having brought the type $\mu x.\mu y.?(y).x$ of Example 5.21 to our attention.

Appendix A. Standard Definitions

The following definitions apply equally well to the language of session types and the language of session contracts. Periodically we change from one to the other in the exposition.

In the language L_{SCts} we have the standard notion of free and bound occurrences of the recursion variables $X, Y, \ldots$ which lead to the standard notion of closed terms.

A.1. **Substitutions.** A substitution is a finite partial map from variables to closed terms. Here we use as an example language that of session contracts, thus for the language L_{SCts} a substitution takes the form

$$\mathbf{s} : \mathsf{var} \rightharpoonup \mathsf{SCts}$$

where $\mathsf{dom}(\mathbf{s})$ is a finite subset of var. Substitutions are composed by letting $\mathbf{s}_1 \cdot \mathbf{s}_2$ have as domain $\mathsf{dom}(\mathbf{s}_1) \cup \mathsf{dom}(\mathbf{s}_2)$, with

$$\mathbf{s}_1 \cdot \mathbf{s}_2(x) = \begin{cases} \mathbf{s}_1(x), & \text{if } x \in \mathsf{dom}(\mathbf{s}_1) \\ \mathbf{s}_2(x), & \text{if } x \in \mathsf{dom}(\mathbf{s}_2),\ x \notin \mathsf{dom}(\mathbf{s}_1) \end{cases}$$

This, together with the empty substitution, denoted ε, endows the collection of substitutions with the structure of a monoid. We use two further operations on substitutions: $\mathbf{s}^{\backslash x}$ has as domain $\mathsf{dom}(\mathbf{s}) - \{x\}$ and acts like $\mathbf{s}$ on all variables in its domain, while for any operator g, $g(\mathbf{s})$ is the substitution with the same domain as $\mathbf{s}$ and maps each X in this set to $g(\mathbf{s}(x))$.

The action of a substitution on a term T, written $T\mathbf{s}$, is defined by structural induction on T. Thus for the language L_{SCts} the definition is as follows:

$$\rho\,\mathbf{s} = \begin{cases} 1 & \text{if } \rho = 1 \\ y & \text{if } \rho = y, y \notin \mathsf{dom}(\mathbf{s}) \\ \mathbf{s}(y) & \text{if } \rho = y, y \in \mathsf{dom}(\mathbf{s}) \\ ?\mathsf{t}.(\rho'\mathbf{s}) & \text{if } \rho = ?\mathsf{t}.\rho' \\ !\mathsf{t}.(\rho'\mathbf{s}) & \text{if } \rho = !\mathsf{t}.\rho' \\ ?(\rho''\mathbf{s}).(\rho'\mathbf{s}) & \text{if } \rho = ?(\rho'').\rho' \\ !(\rho''\mathbf{s}).\rho'\mathbf{s} & \text{if } \rho = !(\rho'').\rho' \\ \mu y.(\rho'\mathbf{s}^{\backslash y}) & \text{if } \rho = \mu y.\rho' \end{cases}$$

Here the important clause is the last one; when applying $\mathbf{s}$ to the recursive term $\mu y.\rho'$ it applies the restricted substitution $\mathbf{s}^{\backslash y}$, which leaves the variable y untouched, to the body ρ'. Note that no notion of α-conversion is required.

It is easy to show the standard compositional property of substitutions, namely $(\rho\,\mathbf{s}_1)\mathbf{s}_2 = \rho\,(\mathbf{s}_1 \cdot \mathbf{s}_2)$, by structural induction on ρ.

$$\frac{S\,[X \mapsto \mu X.S]\ \text{unfold}\ T}{\mu X.S\ \text{unfold}\ T} \qquad\qquad \frac{}{T\ \text{unfold}\ T}\ T \neq \mu X.S$$

Figure 10: Inference rules for the unfolding of terms.

A.2. **Unfoldings.** The unfolding of a closed session term from L_{STyp} is defined *inductively*, as the *least* fixed point of the functional generated by the rules in Figure 10.

It is easy to check that unfold is a function over closed session terms, that is if S unfold T_1 and S unfold T_2 then $T_1 = T_2$. For this reason we use the standard functional notation unfold(S) to denote the unfolding of S. However, this function is not total, as the following example explains.

Example A.1. Observe that there is no finite inference using the rules from Figure 10 that lets us derive an unfolding of $\mu X.X$. The reason is that the last rule in the derivation has to be

$$\frac{\frac{\vdots}{\mu X.X\ \text{unfold}\ T}}{\mu X.X\ \text{unfold}\ T}$$

which leads to a circularity. For this reason unfold($\mu X.X$) is undefined.

It is easy to check also that the function is idempotent, unfold(unfold(S)) = unfold(S). In fact unfold(S) is reminiscent of a *head-normal form*, in the sense that it must take one of the first five forms in the grammar for L_{STyp} in Figure 1.

In order to isolate the terms whose subterms can be unfolded, we use the notion of guarded. Our definition is a mild generalisation of the standard one. Recall that all the constructors but $\mu X.-$ are non-recursive.

Definition A.2. [Guarded]
We say a recursion variable X is guarded in $T \in L_{\text{STyp}}$ if every occurrence of X in T appears under a non-recursive type constructor. Then we say that a term $S \in L_{\text{STyp}}$ is *guarded* if whenever $\mu X.T$ is a subterm of S then X is guarded in T.

Example A.3. Every variable X is a term, and it is not guarded (in itself). In the terms $\mu X.?(X).\text{END}$ and $\mu X.?(\text{END}).X$ the variable X is guarded, for it appears under the constructor $?(-).-$, and so the whole terms are guarded.

Let $T = \mu Y.Y$ and $S = \&\langle 1: T \rangle$. The variable Y is guarded in the term S, for it occurs underneath the type constructor $\&\langle - \rangle$, and it is not guarded in the the term T, because there it occurs directly after the recursive constructor $\mu Y.-$.

Neither S nor T are guarded. The reason is that S is a subterm of itself, and of T, and Y is not guarded in S.

The application of a substitution to a term T in which every variable is guarded preserves the top-most constructor of T. This phenomenon lets us prove the next lemma.

Lemma A.4. *For every $T \in L_{\text{STyp}}$ and substitution $\mathbf{s}$, if every variable in T appears guarded, then* unfold($T\mathbf{s}$) *is defined.*

Proof. The proof is by structural induction on T. Every variable in T appears guarded, thus T cannot be a variable.

The only case worth discussion is when $T = \mu Y.T'$, for in every other case an application of the axiom in Figure 10 ensures the result.

So suppose $T = \mu Y.T'$. Then $T\mathbf{s} = \mu Y.(T'\mathbf{s}^{\backslash Y})$. The hypothesis implies that every variable in T' is guarded, and since T' is a subterm of T, by structural induction we know that $\mathsf{unfold}(T'\mathbf{s}')$ is defined for every substitution $\mathbf{s}'$.

Let us pick the particular substitution $\mathbf{s}' = \mathbf{s}^{\backslash Y} \cdot \mathbf{s}''$, where $\mathbf{s}''$ maps the variable Y to $T\mathbf{s}$. We know by induction that we can infer

$$T'(\mathbf{s}^{\backslash Y} \cdot \mathbf{s}'') \; \mathsf{unfold} \; R$$

for some R. But by compositionality we know $T'(\mathbf{s}^{\backslash Y} \cdot \mathbf{s}'') = (T'\mathbf{s}^{\backslash Y})\mathbf{s}''$ and so one application of the rule on the right in Figure 10 gives the required

$$T\mathbf{s} \; \mathsf{unfold} \; R$$

$\square$

Lemma A.5. *If T is a guarded closed term in L_{STyp} then $\mathsf{unfold}(T)$ is defined.*

Proof. For every substitution $\mathbf{s}$, since T is closed, $T\mathbf{s} = T$. Lemma A.4 ensures that $\mathsf{unfold}(T\mathbf{s})$ is defined, so $\mathsf{unfold}(T)$ is defined. $\square$

The main properties used of the unfold function are now collected in the following lemma:

Lemma A.6. *For every session contract $\sigma \in \mathsf{SCts}$,*

(a) *if $\sigma \xrightarrow{\tau} {}^{\star} \sigma'$ and σ' is stable, then $\mathsf{unfold}(\sigma) \xrightarrow{\tau} {}^{\star} \sigma'$.*
(b) *if σ is m-closed then so is $\mathsf{unfold}(\sigma)$.*

Proof. Recall that all terms in SCts are closed and guarded, and therefore by the previous lemma, applied to session contracts, $\mathsf{unfold}(\sigma)$ is defined. For convenience let it be denoted by ρ.

The proof of (a) now proceeds by induction on the derivation of $\sigma \; \mathsf{unfold} \; \rho$ from the rules in Figure 10, and a case analysis on the structure of σ. The only non-trivial case is when σ has the form $\mu x.\sigma_1$.

Since σ' is stable and there is exactly one rule from Figure 3 which can apply to $\mu x.\sigma_1$, the sequence of transitions $\sigma \xrightarrow{\tau} {}^{\star} \sigma'$ must actually take the form

$$\mu x.\sigma_1 \xrightarrow{\tau} \sigma_1[x \mapsto \sigma] \xrightarrow{\tau} {}^{\star} \sigma'$$

By induction $\mathsf{unfold}(\sigma_1[x \mapsto \sigma]) \xrightarrow{\tau} {}^{\star} \sigma'$. The result now follows since by the rules of Figure 3 we know $\mathsf{unfold}(\rho[x \mapsto \sigma])$ coincides with ρ.

The proof of statement (b) has a similar structure. $\square$

Note that in the statement of Lemma A.6(a) the hypothesis that σ' be stable is essential. As a counterexample consider the case when σ is $\mu x.\mu y.!\mathbf{1}.x$. We have $\sigma \xrightarrow{\tau} {}^{\star} \mu y.!\mathbf{1}.\sigma$ but $\mathsf{unfold}(\sigma) \not\xrightarrow{\tau} {}^{\star} \mu y.!\mathbf{1}.\sigma$, since $\mathsf{unfold}(\sigma)$ is $!\mathbf{1}.\sigma$.

Appendix B. Type derivations

In this appendix we discuss the derivation trees needed to prove the typing judgement $\vdash P$ and $\vdash Q$, where P and Q are the processes respectively of Example 1.2 and Example 5.22.

Recall the type equivalence $=_{\mathsf{eq}}$ defined in Section 4 (4.2). To use the typing rules of [YV07] and [Vas12] we reason up-to type equivalence.

We discuss first the derivation of $\vdash P$, which is shown in In Figure 11. The rules we used are given in [YV07, pag. 89], and the ones not defined there are defined in Figure 6 of the same paper. The only difference between our presentation and [YV07] is that in rule [CRes] we use a relation $\mathcal{D}$

$$\dfrac{\dfrac{y\colon \text{END} \quad \text{completed}}{X\colon T_x, T_y \vdash X\langle z, \kappa_{\mathbf{f}}^- \rangle \rhd y\colon \text{END},\ z\colon T_z,\ \kappa_{\mathbf{f}}^-\colon T_{\mathbf{f}}^-}\ [\text{Var}]^C}{X\colon T_x, T_y \vdash \mathtt{catch}\ y(z)\ \mathtt{in}\ X\langle z, \kappa_{\mathbf{f}}^- \rangle \rhd y\colon ?(T_z).\text{END},\ \kappa_{\mathbf{f}}^-\colon T_{\mathbf{f}}^-}\ [\text{Cat}]^B$$

$$\dfrac{\dfrac{\dfrac{x\colon \text{END} \quad \text{completed}}{X\colon T_x, T_y \vdash \mathbf{0} \rhd x\colon \text{END}}\ [\text{Inact}]}{X\colon T_x, T_y \vdash \mathtt{throw}\ x[\kappa_{\mathbf{f}}^+];\mathbf{0} \rhd x\colon\ !(T_{\mathbf{f}}^+).\text{END},\ \kappa_{\mathbf{f}}^+\colon T_{\mathbf{f}}^+}\ [\text{Thr}]^A \qquad \dfrac{\vdots}{(**)}}{\begin{array}{c}X\colon T_x, T_y \vdash \mathtt{throw}\ x[\kappa_{\mathbf{f}}^-];\mathbf{0}\ \| \\ \mathtt{catch}\ y(z)\ \mathtt{in}\ X\langle z, \kappa_{\mathbf{f}}^- \rangle \rhd x\colon T_x,\ y\colon T_y,\ \kappa_{\mathbf{f}}^-\colon T_{\mathbf{f}}^-,\ \kappa_{\mathbf{f}}^+\colon T_{\mathbf{f}}^+\end{array}}\ [\text{Conc}]}{\begin{array}{c}X\colon T_x, T_y \vdash (\nu\kappa_{\mathbf{f}})(\mathtt{throw}\ x[\kappa_{\mathbf{f}}^-];\mathbf{0}\ \| \\ \mathtt{catch}\ y(z)\ \mathtt{in}\ X\langle z, \kappa_{\mathbf{f}}^- \rangle) \rhd x\colon T_x,\ y\colon T_y\end{array}}\ T_{\mathbf{f}}^+\ \mathcal{D}\ T_{\mathbf{f}}^-;\ [\text{CRes}]$$

$$\dfrac{\dfrac{\dfrac{\vdots}{(*)} \qquad \dfrac{\emptyset \quad \text{completed}}{X\colon T_x, T_y \vdash X\langle \kappa_{\mathbf{o}}^+, \kappa_{\mathbf{o}}^- \rangle \rhd \kappa_{\mathbf{o}}^+\colon T_x,\ \kappa_{\mathbf{o}}^-\colon T_y}\ [\text{Var}]}{\vdash \mathtt{def}\ D\ \mathtt{in}\ X\langle \kappa_{\mathbf{o}}^+, \kappa_{\mathbf{o}}^- \rangle \rhd \kappa_{\mathbf{o}}^+\colon T_x,\ \kappa_{\mathbf{o}}^-\colon T_y}\ [\text{Def}]}{\vdash (\nu\kappa_{\mathbf{o}})(\mathtt{def}\ D\ \mathtt{in}\ X\langle \kappa_{\mathbf{o}}^+, \kappa_{\mathbf{o}}^- \rangle)}\ T_x\ \mathcal{D}\ T_y;\ [\text{CRes}]$$

$$
\begin{array}{ll}
A) & T_x =_{\mathsf{eq}}\ !(T_{\mathbf{f}}^+).\text{END} \\[2pt]
B) & T_y =_{\mathsf{eq}}\ ?(T_z).\text{END} \\[2pt]
C) & T_x =_{\mathsf{eq}} T_z, \quad T_y =_{\mathsf{eq}} T_{\mathbf{f}}^-
\end{array}
$$

Figure 11: Derivation tree to infer $\vdash P$ in the type system of [YV07], but with rule [CRes] of our
Figure 7

instead of the duality $\overline{}$. This allows us to compare the typability of P using prdual and $\overline{}$ as duality
relations.

The existence of the derivation tree depends on the satisfaction of the constraints that we gather
building the tree. In Figure 11 the constraints and the rule applications that generate them are labelled
respectively with A, B, C.

We show how to satisfy the conditions, which are the following ones.

$$
\begin{array}{llll}
T_x & =_{\mathsf{eq}} & !(T_{\mathbf{f}}^+).\text{END} & \qquad T_x \quad =_{\mathsf{eq}} \quad T_z \\[4pt]
T_y & =_{\mathsf{eq}} & ?(T_z).\text{END} & \qquad T_y \quad =_{\mathsf{eq}} \quad T_{\mathbf{f}}^- \\[4pt]
T_x & \mathcal{D} & T_y & \qquad T_{\mathbf{f}}^+ \quad \mathcal{D} \quad T_{\mathbf{f}}^-
\end{array}
$$

We let T_z and $T_{\mathbf{f}}^-$ to be definitionally equal to, respectively, T_x and T_y. In view of these definitions,
the constraints we have to satisfy are the following ones,

$$
\begin{array}{llll}
T_x & =_{\mathsf{eq}} & !(T_{\mathbf{f}}^+).\text{END} & \qquad T_x \quad \mathcal{D} \quad T_y \\[4pt]
T_y & =_{\mathsf{eq}} & ?(T_x).\text{END} & \qquad T_{\mathbf{f}}^+ \quad \mathcal{D} \quad T_y
\end{array}
$$

Given the condition that $T_x \mathcal{D} T_y$ an obvious way to satisfy $T_{\mathbf{f}}^+ \mathcal{D} T_y$ is to let $T_{\mathbf{f}}^+$ be definitionally T_x. This leaves us with three conditions

$$T_x =_{\mathsf{eq}} !(T_x).\textsc{end}, \quad T_y =_{\mathsf{eq}} ?(T_x).\textsc{end}, \quad T_x \mathcal{D} T_y$$

The two equations are solved by letting $T_x = \mu X.!(X).\textsc{end}$, and $T_y = ?(\mu X.!(X).\textsc{end}).\textsc{end}$. If we take $\mathcal{D}$ to be $\bar{\cdot}$, then the types T_x and T_y cannot satisfy $T_x \mathcal{D} T_y$, because $\overline{T_x} \neq_{\mathsf{eq}} T_y$. If we take $\mathcal{D}$ to be prdual, then the condition on $\mathcal{D}$ is satisfied, because $\mathsf{cplmt}(T_x) =_{\mathsf{eq}} T_y$.

Now we discuss the derivation of $\vdash Q$, where

$$Q = (\nu xy)\,(\,\overline{x}\langle\,x\,\rangle.\mathbf{0} \parallel \mathsf{un}\, y(\,z\,).\overline{z}\langle\,z\,\rangle.\mathbf{0}\,)$$

is the process we already used in Example 5.22. We show the derivation tree in Figure 12. Let us adapt our notions to the setting of [Vas12]. We use the the syntax given in Figure 3 of that paper,

q	::=		**Qualifiers**
		lin	linear
		un	unrestricted
t	::=		**Types**
		Bool	boolean
		END	termination
		(q, T)	qualified pretype

where our terms $T, S, \ldots$ are considered *pretypes*, and a type t is pair composed by a qualifier and a pretype. Now we lift the relation $=_{\mathsf{eq}}$ (i.e. the equivalence due to $\preccurlyeq$) to types by writing $(q, T) =_{\mathsf{eq}} (q', T')$ whenever $q = q'$ and $T =_{\mathsf{eq}} T'$. We also lift the duality function $\mathsf{prdual}(-)$ to types, and let

$$\mathsf{prdualQ}(q, T) = (q, \mathsf{prdual}(T)) \tag{2.1}$$

This definition is analogous to the one in [Vas12, Figure 4], but there the standard duality is employed.

The premises of the rules used in the derivation tree of Figure 12 ensure that $T_y =_{\mathsf{eq}} \mathsf{un}\,?(T_z).\textsc{end}$, and the two equations A) and B) in that figure have the same solution, so $T_z \preccurlyeq T_x$. It turn this implies that $T_y =_{\mathsf{eq}} \mathsf{un}\,?(T_x).\textsc{end}$. The existence of the derivation tree depends on two conditions, namely

$$T_x =_{\mathsf{eq}} \mathsf{un}\,!(T_x).\textsc{end}, \quad T_x \mathcal{D} T_y$$

We already know that the equation is satisfied by the type $\hat{T} = \mathsf{un}\,\mu X.!(X).\textsc{end}$, so we have just to find a $\mathcal{D}$ such that $\hat{T} \mathcal{D} T_y$. If we instantiate $\mathcal{D}$ to the function $\mathsf{prdualQ}$ defined (2.1) above, then the condition $\hat{T} \mathcal{D} T_y$ is satisfied. If we instantiate $\mathcal{D}$ to the standard duality, then that condition is not satisfied, so a derivation tree necessary to conclude $\vdash Q$ does not exist.

References

[AN01]   A. Arnold and D. Niwiński. *Rudiments of μ-calculus*. Studies in Logic and the Foundations of Mathematics. Elsevier, 2001.

[Bd10]   Franco Barbanera and Ugo de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In Temur Kutsia, Wolfgang Schreiner, and Maribel Fernández, editors, *PPDP*, pages 155–164. ACM, 2010.

[BdL13] Franco Barbanera and Ugo de' Liguoro. Sub-behaviour relations for session-based client/server systems. submitted for publication, 2013.

[Ber13] Giovanni Bernardi. *Behavioural Equivalences for Web Services*. PhD thesis, Trinity College Dublin, 2013. Available at https://www.scss.tcd.ie/~bernargi.

[BH98]   Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. *Fundam. Inform.*, 33(4):309–338, 1998.

$$\dfrac{\overline{z\colon T_z \vdash z\colon \mathsf{un}\,!(T_z).\mathrm{END} \quad z\colon T_z \vdash z\colon T_z \quad \overline{y\colon \mathrm{END},\ z\colon \mathrm{END} \vdash \mathbf{0}}}\;[\text{T-Inact}]}{y\colon \mathrm{END},\ z\colon T_z \vdash \overline{z}\langle z \rangle.\mathbf{0}}\;[\text{T-Out}]^B$$

$$\dfrac{\mathsf{un}\,(y\colon \mathsf{un}\,?(T_z).\mathrm{END}) \quad y\colon \mathsf{un}\,?(T_z).\mathrm{END} \vdash y\colon \mathsf{un}\,?(T_z).\mathrm{END} \quad \overset{\vdots}{(\ast\ast)}}{y\colon T_y \vdash \mathsf{un}\,y(z).\overline{z}\langle z \rangle.\mathbf{0}}\;[\text{T-In}]$$

$$\dfrac{\dfrac{x\colon T_x \vdash x\colon \mathsf{un}\,!(T_x).\mathrm{END} \quad x\colon T_x \vdash x\colon T_x \quad \overline{x\colon \mathrm{END} \vdash \mathbf{0}}\;[\text{T-Inact}]}{x\colon T_x \vdash \overline{x}\langle x \rangle.\mathbf{0}}\;[\text{T-Out}]^A \quad \overset{\vdots}{(\ast)}}{\dfrac{x\colon T_x,\ y\colon T_y \vdash \overline{x}\langle x \rangle.\mathbf{0} \parallel \mathsf{un}\,y(z).\overline{z}\langle z \rangle.\mathbf{0}}{\vdash (\nu xy)\,(\overline{x}\langle x \rangle.\mathbf{0} \parallel \mathsf{un}\,y(z).\overline{z}\langle z \rangle.\mathbf{0})}\;[\text{T-Res}] \quad T_x\,\mathcal{D}\,T_y}\;[\text{T-Par}]$$

$$A) \quad T_x =_{\mathsf{eq}} !(T_x).\mathrm{END}$$
$$B) \quad T_z =_{\mathsf{eq}} !(T_z).\mathrm{END}$$

Figure 12: Derivation tree to infer $\vdash Q$ in the type system of [Vas12], but with rule [T-Res] of our Figure 8

[BH12]   Giovanni Bernardi and Matthew Hennessy. Modelling session types using contracts. In Sascha Ossowski and Paola Lecca, editors, *SAC*, pages 1941–1946. ACM, 2012.

[BH14]   Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types (extended abstract). In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2014.

[BP12]   Viviana Bono and Luca Padovani. Typing copyless message passing. *Logical Methods in Computer Science*, 8(1), 2012.

[CCLP06]   Samuele Carpineti, Giuseppe Castagna, Cosimo Laneve, and Luca Padovani. A formal account of contracts for web services. In Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro, editors, *WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2006.

[CDCGP09] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In António Porto and Francisco Javier López-Fraguas, editors, *PPDP*, pages 219–230. ACM, 2009.

[CDV08]   Giuseppe Castagna, Rocco De Nicola, and Daniele Varacca. Semantic subtyping for the pi-calculus. *Theor. Comput. Sci.*, 398(1-3):217–242, 2008.

[CGP09]   Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM Trans. Program. Lang. Syst.*, 31(5):1–61, 2009. Supersedes the article in POPL '08.

[DCd09]   Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. Sessions and session types: An overview. In Cosimo Laneve and Jianwen Su, editors, *WS-FM*, volume 6194 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2009.

[DCdY07]   Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Nobuko Yoshida. On progress for structured communications. In Gilles Barthe and Cédric Fournet, editors, *TGC*, volume 4912 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2007.

[DH84]   Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[DH11]   Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2011.

[DY13]   Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska,

and David Peleg, editors, *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2013.

[FCB08]   Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *J. ACM*, 55(4):19:1–19:64, September 2008.

[FV13]   Juliana Franco and Vasco T. Vasconcelos. A concurrent programming language with refined session types. In *Second International Workshop on Behavioural Types*, pages 33–42. University Complutense of Madrid, 2013.

[GH05]   Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005.

[HMM⁺12]   Kohei Honda, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Verification of mpi programs using session types. In Jesper Larsson Träff, Siegfried Benkner, and Jack J. Dongarra, editors, *EuroMPI*, volume 7490 of *Lecture Notes in Computer Science*, pages 291–293. Springer, 2012.

[Hoa85]   C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, 1985.

[HVK98]   Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *ESOP*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998.

[LP07]   Cosimo Laneve and Luca Padovani. The must preorder revisited. In *Proceedings of the 18th international conference on Concurrency Theory*, pages 212–225, Berlin, Heidelberg, 2007. Springer-Verlag.

[LP08]   Cosimo Laneve and Luca Padovani. The pairing of contracts and session types. In Pierpaolo Degano, Rocco De Nicola, and José Meseguer, editors, *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2008.

[Mil89a]   R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Mil89b]   Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[Pad10]   Luca Padovani. Contract-based discovery of web services modulo simple orchestrators. *Theor. Comput. Sci.*, 411(37):3328–3347, 2010.

[Pad11]   Luca Padovani. Fair Subtyping for Multi-Party Session Types. In *Proceedings of the 13th Conference on Coordination Models and Languages*, volume LNCS 6721, pages 127–141. Springer, 2011.

[PS96]   Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[Tar55]   Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5(2):285–309, 1955.

[THK94]   Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In Constantine Halatsis, Dimitris G. Maritsas, George Philokyprou, and Sergios Theodoridis, editors, *PARLE*, volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer, 1994.

[Vas12]   Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012.

[Vas13]   Vasco Thudichum Vasconcelos, 2013. Private communication.

[VV13]   Hugo Torres Vieira and Vasco Thudichum Vasconcelos. Typing progress in communication-centred systems. In Rocco De Nicola and Christine Julien, editors, *COORDINATION*, volume 7890 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 2013.

[Wad12]   Philip Wadler. Propositions as sessions. In Peter Thiemann and Robby Bruce Findler, editors, *ICFP*, pages 273–286. ACM, 2012.

[YV07]   Nobuko Yoshida and Vasco Thudichum Vasconcelos. 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.