On duality relations for session types

Giovanni Bernardi[1], Ornela Dardha[2], Simon J. Gay[2], and Dimitrios Kouzapas[2,3]

[1] IMDEA Software Institute, Madrid, Spain
bernargi@tcd.ie
[2] School of Computing Science, University of Glasgow, UK
{Ornela.Dardha, Simon.Gay, Dimitrios.Kouzapas}@glasgow.ac.uk
[3] Department of Computing, Imperial College London, UK

Abstract. Session types are a type formalism used to describe communication protocols over private session channels. Each participant in a binary session owns one endpoint of a session channel. A key notion is that of *duality*: the endpoints of a session channel should have dual session types in order to guarantee communication safety. Duality relations have been independently defined in different ways and different works, without considering their effect on the type system. In this paper we systematically study the existing duality relations and some new ones, and compare them in order to understand their expressiveness. The outcome is that those relations are split into two groups, one related to the naïve inductive duality, and the other related to a notion of mutual compliance, which we borrow from the literature on contracts for web-services.

1 Introduction

A *session* is a private connection between a finite number of participants, and a *binary* session involves exactly two participants. A binary session can be thought of as a private communication channel, say a, with two *endpoints*, respectively a^+ and a^-. Each participant in a binary session owns one of the session endpoints and uses it to communicate with the other participant.

Session types are type-theoretic specifications of communication protocols. They can be incorporated into a programming language type system so that the correctness of the communication operations in a program can be verified by static type checking. In this paper we focus on types for binary sessions, that is binary session types [12].

A typical formalism to study binary sessions is the π-calculus, where session channels are identified by private names $a, b, c, \ldots$, and their endpoints are distinguished syntactically, $a^+, a^-, \ldots$. For instance, consider two agents A and B which communicate over a session c according to the following protocol: A sends an integer to B and then receives a boolean value from B. In the π-calculus with session types this is written as

$$(\nu c\colon S)\,(\,c^+![4].c^+?[v\colon \mathsf{bool}].\mathbf{0} \mid c^-?[n\colon \mathsf{int}].c^-![\mathsf{false}].\mathbf{0}\,)$$

where the protocol for A is described by the session type $S = ![\mathsf{int}];?[\mathsf{bool}];\text{END}$. This type describes only half of the overall protocol, namely what takes place in c^+. The other half of the protocol takes place in c^-, and is described by the session type $T = ?[\mathsf{int}];![\mathsf{bool}];\text{END}$ which models the protocol for agent B.

In the syntax of types, output is denoted by !, input is denoted by ?, and END denotes the type of the successfully terminated session. The ";" models the continuation of the session type. Moreover, it is possible to define types with a branching structure in which one agent selects from a set of possibilities offered by the other agent.

A session type system checks that *i*) the communication operations in A match the type S; *ii*) the communication operations in B match the type T; and *iii*) S and T are related by an appropriate *duality* relation: $S \ \mathcal{D} \ T$, which intuitively describes opposite behaviours between two communicating agents. The above three steps ensure that communications within sessions are error-free.

The duality relation is a distinctive feature of session type systems, with no clear analogue in, for example, λ-calculus type systems. Recently Bernardi and Hennessy [2] have shown that duality relations have a non-trivial impact on the type systems, therefore should be carefully defined. This is the case in particular in the presence of non-tail recursive session types, like $\mu X.\,![\,X\,].\text{END}$. The following example exhibits a safe process that uses the aforementioned non-tail recursive session type.

Example 1. Consider the following processes,

$$P = (\nu t\colon \sharp T_b)\,(\nu u\colon \sharp S)\,((\nu b\colon T_b)\,(\,t![b^+].u![b^-].\mathbf{0}\,)\mid !R\,)$$
$$R = t?[x\colon T_x].u?[y\colon T_y].(\nu a\colon T_a)\,(\,x![a^+].\mathbf{0}\mid y?[z\colon T_z].t![z].u![a^-].\mathbf{0}\,)$$

where $!R$ denotes the replication of R.

Intuitively, process P is safe because its reductions never lead to communication errors. By following the type system presented in the next section, the type derivation tree for P requires the type T_a to be $\mu X.\,![\,X\,].\text{END}$, which is non-tail recursive (see [2, Section 5.2]). □

Beyond the obvious point that duality should reverse the direction of messages, existing papers on session types do not give explicit justification for the exact way in which they define duality, and different authors follow different intuitions.

The aim of this paper is to study the existing duality relations in a systematic way; we contrast and compare a number of these relations as follows.

First, we compare the duality relations in a set-theoretic way, and we show that they can be classified in two groups, one related to a symmetric notion of testing, called "peer compliance", and the other related to the standard inductive duality defined by Honda *et al.* [12].

Second, we show that the type system of Gay and Hole [10] is robust with respect to the dualities in the peer compliance group, in the sense that its typing relation is the same for all *syntactic* duality relations in that group. This means that type-checkers implementing the type system can use the smallest duality in the peer compliance group.

Structure of the paper: In Section 2 we define the π-calculus with sessions based on [10]. It presents the statics and the dynamics of the calculus. We then consider several syntax-oriented definitions of duality in the presence of recursive types. In Section 3 we recall the behavioural theory of session types defined in [2]. In Section 4 we make a set-theoretic comparison of all the duality relations, establish a hierarchy with respect to inclusion, and consider the duality relation as a parameter of the type system. We discuss related work in Section 5, and conclude the paper in Section 6.

2 The π-calculus with sessions

In this section we introduce the π-calculus with session types: the syntax of types and terms, the operational semantics and the typing discipline which enjoys type preservation and type safety [10], thus guaranteeing error-free communications.

2.1 Session Types

Let I, J, K range over $\mathcal{I} = \{ X \subseteq \mathbb{N} \mid X \text{ finite, non-empty} \}$, and let L_{SType} be the language of terms defined by the grammar (a):

$$(a) \quad S, T ::= \text{END} \mid X \mid ?[\,T\,].S \mid ![\,T\,].S \mid \&\langle l_i : S_i \rangle_{i\in I} \mid \oplus\langle l_i : S_i \rangle_{i\in I} \mid \mu X.S$$

$$(b) \quad M, N ::= S \mid \sharp M \mid X \mid \mu X.M$$

where the labels l_i in terms are pairwise distinct. Let *SType* be the set of session type terms which are *closed* (they contain no free type variables) and *guarded* (they contain a non-recursive type constructor between every type variable and its μ binder). We refer to the terms in *SType* as *session types* [2, Appendix A].

Type END is the type of a terminated session channel; $?[\,T\,].S$ and $![\,T\,].S$ indicate respectively the types for receiving and sending a message of type T and continuation of type S. Branch and select, $\&\langle l_i : S_i \rangle_{i\in I}$ and $\oplus\langle l_i : S_i \rangle_{i\in I}$ are sets of labelled session types indicating, respectively, external and internal choice. X and $\mu X.S$ model recursive session types.

The language of type terms is given by the grammar (b) above.We refer to the set of closed and guarded type terms as *types*. A type can be a session type, a standard shared channel type, a type variable or a recursive type.

In the present paper session types are higher-order, and their messages do not contain shared channel types. We use the pure higher-order session types approach to respect the semantic model defined in later Section 3.

Usually, the typing discipline of session types includes a duality *function* which constructs a specific dual type for any given session type. The following is the definition of inductive duality [12].

Definition 1 (Inductive Duality). *The inductive duality function is defined as:*

$$\overline{X} = X \qquad \overline{?[\,T\,].S} = ![\,T\,].\overline{S} \qquad \overline{\&\langle l_i : S_i \rangle_{i\in I}} = \oplus\langle l_i : \overline{S_i} \rangle_{i\in I} \qquad \overline{\mu X.S} = \mu X.\overline{S}$$
$$\overline{\text{END}} = \text{END} \qquad \overline{![\,T\,].S} = ?[\,T\,].\overline{S} \qquad \overline{\oplus\langle l_i : S_i \rangle_{i\in I}} = \&\langle l_i : \overline{S_i} \rangle_{i\in I}$$

□

Although standard and broadly used, in [2, 3] it was observed that inductive duality is not adequate in the presence of recursive session types, as it does not commute with the unfolding of non-tail-recursive types such as $\mu X.\,![\,X\,].\text{END}$, which in general are necessary (see Example 1 in the Introduction).

In order to overcome this inadequacy, Bono and Padovani [3], and Bernardi and Hennessy [2] independently defined an alternative function, called by the latter authors *complement*, and here denoted as cplt. In the following we give the definition of the complement function on session type terms, by adapting the corresponding one in [2].

Definition 2 (Complement Function). *The function* $\mathsf{cplt} : L_{SType} \longrightarrow L_{SType}$ *is defined as:*

$$\begin{array}{llll} \mathsf{cplt}(?[\,T\,].S) & = ![\,T\,].\mathsf{cplt}(S) & \mathsf{cplt}(X) & = X \\ \mathsf{cplt}(![\,T\,].S) & = ?[\,T\,].\mathsf{cplt}(S) & \mathsf{cplt}(\text{END}) & = \text{END} \\ \mathsf{cplt}(\&\langle\, l_i : S_i \,\rangle_{i\in I}) & = \oplus\langle\, l_i : \mathsf{cplt}(S_i) \,\rangle_{i\in I} & \mathsf{cplt}(\mu X.S) & = \mu X.\mathsf{cplt}(S\lfloor^{\mu X.S}/_X\rfloor) \\ \mathsf{cplt}(\oplus\langle\, l_i : S_i \,\rangle_{i\in I}) & = \&\langle\, l_i : \mathsf{cplt}(S_i) \,\rangle_{i\in I} & & \end{array}$$ □

The complement relies on a syntactic substitution $\lfloor^{-}/_{-}\rfloor$, which acts only on the carried types. The details are in [2, Section 5] and the formal definition is as follows,

$$\begin{array}{llll} ?[\,T\,].S'\lfloor^S/_X\rfloor & = ?[\,T\left\{^S/_X\right\}\,].(S'\lfloor^S/_X\rfloor) & Y\lfloor^S/_X\rfloor & = Y \\ ![\,T\,].S'\lfloor^S/_X\rfloor & = ![\,T\left\{^S/_X\right\}\,].(S'\lfloor^S/_X\rfloor) & \text{END}\lfloor^S/_X\rfloor & = \text{END} \\ \&\langle\, l_i : S_i \,\rangle_{i\in I}\lfloor^S/_X\rfloor & = \&\langle\, l_i : S_i\lfloor^S/_X\rfloor \,\rangle_{i\in I} & (\mu Y.S')\lfloor^S/_X\rfloor & = \mu Y.(S'\lfloor^S/_X\rfloor) \quad \text{if } Y \neq X \\ \oplus\langle\, l_i : S_i \,\rangle_{i\in I}\lfloor^S/_X\rfloor & = \oplus\langle\, l_i : S_i\lfloor^S/_X\rfloor \,\rangle_{i\in I} & (\mu X.S')\lfloor^S/_X\rfloor & = \mu X.S' \end{array}$$

In the presence of recursive types and their unfoldings, another standard approach is to define duality to be a *relation* rather than a function. For instance, $\mu X.?\,[\mathsf{int}]; X$ and $!\,[\mathsf{int}]; \mu X.!\,[\mathsf{int}]; X$ model dual behaviours and should be considered dual types. However, this duality is not captured by the complement function. For this reason we define in the following the *co-inductive duality* relation [10]. In order to define co-inductive duality, we first have to define unfolding of terms and the subtyping relation.

The unfolding function UNF unfolds a recursive type until the first type constructor different from $\mu X.-$ is reached. Formally, UNF is defined on syntactic terms as the smallest relation that satisfies the following inference rules:

$$\frac{S\left\{^{\mu X.S}/_X\right\} \text{ UNF } S'}{\mu X.S \text{ UNF } S'} \qquad \frac{}{S \text{ UNF } S}\ S \neq \mu X.S'$$

The relation UNF is by definition a partial function on terms of the language L_{SType}, and it is defined on every closed and guarded term [2, Lemma A.8].

Definition 3 (Subtyping). *Let* $\mathcal{F} : \mathcal{P}(SType^2) \longrightarrow \mathcal{P}(SType^2)$ *be the functional defined so that* $(T, S) \in \mathcal{F}(\mathcal{R})$ *whenever all the following conditions hold:*

(i) *if* $\text{UNF}(T) = \text{END}$ *then* $\text{UNF}(S) = \text{END}$
(ii) *if* $\text{UNF}(T) = ![\,T_m\,].T'$ *then* $\text{UNF}(S) = ![\,S_m\,].S'$ *and* $S_m\ \mathcal{R}\ T_m$ *and* $T'\ \mathcal{R}\ S'$
(iii) *if* $\text{UNF}(T) = ?[\,T_m\,].T'$ *then* $\text{UNF}(S) = ?[\,S_m\,].S'$ *and* $T_m\ \mathcal{R}\ S_m$ *and* $T'\ \mathcal{R}\ S'$
(iv) *if* $\text{UNF}(T) = \oplus\langle\, l_i : T_i \,\rangle_{i\in I}$ *then* $\text{UNF}(S) = \oplus\langle\, l_j : S_j \,\rangle_{j\in J}$, $J \subseteq I$, $T_j\ \mathcal{R}\ S_j$ *for all* $j \in J$
(v) *if* $\text{UNF}(T) = \&\langle\, l_i : T_i \,\rangle_{i\in I}$ *then* $\text{UNF}(S) = \&\langle\, l_j : S_j \,\rangle_{j\in J}$, $I \subseteq J$, $T_i\ \mathcal{R}\ S_i$ *for all* $i \in I$

If $\mathcal{R} \subseteq \mathcal{F}(\mathcal{R})$*, then we say that* $\mathcal{R}$ *is a* type simulation*. Standard arguments ensure that there exists the greatest solution of the equation* $X = \mathcal{F}(X)$*. Let* $\preccurlyeq_{\mathsf{sbt}} = \nu X.\mathcal{F}(X)$*, and let* $=_{\mathsf{sbt}}$ *be the equivalence generated by* $\preccurlyeq_{\mathsf{sbt}}$*. We call* $\preccurlyeq_{\mathsf{sbt}}$ *the* subtyping relation. □

Observe that the relation $\preccurlyeq_{\mathsf{sbt}}$ defined above is less general than the original one in [10], in that it is not defined on channel types, and less general than the one in [2] in that here we do not have base types. Both are insignificant restrictions since subtyping on standard channel types and base types can be easily integrated into our framework.

Definition 4 (Co-inductive Duality, Syntactic Compliance). *Let* $\mathcal{G} : \mathcal{P}(SType^2) \times \mathcal{P}(SType^2) \times \mathcal{P}(\mathcal{I}^2) \longrightarrow \mathcal{P}(SType^2)$ *be the functional defined so that* $(T, \mathcal{S}) \in \mathcal{G}(\mathcal{R}, \mathcal{B}, C)$ *whenever all the following conditions hold:*

(i) *If* $\text{UNF}(T) = \text{END}$ *then* $\text{UNF}(S) = \text{END}$
(ii) *If* $\text{UNF}(T) = \,?\,[T_m]; T'$ *then* $\text{UNF}(S) = \,!\,[S_m]; S'$, $T' \,\mathcal{R}\, S'$, *and* $T_m \,\mathcal{B}\, S_m$
(iii) *If* $\text{UNF}(T) = \,!\,[T_m]; T'$ *then* $\text{UNF}(S) = \,?\,[S_m]; S'$, $T' \,\mathcal{R}\, S'$, *and* $T_m \,\mathcal{B}\, S_m$
(iv) *If* $\text{UNF}(T) = \oplus\langle\, l_i : T_i \,\rangle_{i\in I}$ *then* $\text{UNF}(S) = \&\langle\, l_j : S_j \,\rangle_{j\in J}$, $I \,C\, J$, $T_i \,\mathcal{R}\, S_i$ *for all* $i \in I$
(v) *If* $\text{UNF}(T) = \&\langle\, l_i : T_i \,\rangle_{i\in I}$ *then* $\text{UNF}(S) = \oplus\langle\, l_j : S_j \,\rangle_{j\in J}$, $J \,C\, I$, $S_j \,\mathcal{R}\, T_j$ *for all* $j \in J$

Standard techniques ensure that for every $\mathcal{B}$ *and* C *there exists the greatest solution of the equation* $X = \mathcal{G}(X, \mathcal{B}, C)$. *We let* $\bot_{\mathsf{c}} = \nu X.\mathcal{G}(X, =_{\mathsf{sbt}}, =)$, *and let* $\mathtt{dual}^{\mathtt{sbt}} = \nu X.\mathcal{G}(X, \preccurlyeq_{\mathsf{sbt}}, \subseteq)$. *We call* $\bot_{\mathsf{c}}$ coinductive duality *and* $\mathtt{dual}^{\mathtt{sbt}}$ syntactic compliance. □

In Definition 3 and Definition 4 we treat recursion explicitly, because we do not take the widespread approach whereby (⋆) "a type can be freely replaced by its unfolding". In general assuming (⋆) is inconsistent, as shown in the following example.

Example 2. Here we prove a false equality by assuming (⋆) and using the function #–height(T) of [13], which counts the number of top-most recursive constructors in T. For instance, the equalities (a) #–height($\mu X.\text{END}$) = 1, (b) #–height(END) = 0 and (c) #–height($\mu X.\text{END}$) = #–height($\mu X.\text{END}$) are true. Thanks to (⋆), in the right-hand side of (c) we replace $\mu X.\text{END}$ with END, thereby obtaining #–height($\mu X.\text{END}$) = #–height(END). Now (a) and (b) imply the equality 1 = 0, which is false. □

2.2 Session Processes

Our language is the π-calculus with session types, defined in [10]. We do not allow standard channels as messages, but this restriction is enforced by the syntax of types, not by modifications to the language. We also restrict messages to be monadic, for notational convenience and to match the semantic model of session types in Section 3. The syntax of session processes is defined by the following grammar:

$$P, Q ::= \mathbf{0} \mid (P \mid Q) \mid\, !P \mid x^p?[y\colon M].P \mid x^p![y^p].P \mid (\nu x\colon M)\, P \mid x \triangleright \{l_i : P_i\}_{i\in I} \mid x \triangleleft l.P$$

Let $x, y, z \ldots$ range over names, and $l_1, l_2, \ldots$ over labels. Names may be *polarised* in order to distinguish between the endpoints of a channel, occurring as x^+ or x^- or simply as x. We write x^p for a general polarised name, where p is +, −, or empty. Duality on polarities, written $\overline{p}$, exchanges + and −. Process $(\nu x{:}T)P$ binds x in P and process $x^p\,?\,[y{:}T]; P$ binds y in P. In $(\nu x{:}T)P$, both x^+ and x^- may occur in P, and both are bound. In $x^p\,?\,[y{:}T]; P$, only y (unpolarised) may occur in P. If a name is not bound in P then it is free in P. FN(P) denotes the set of free names in P. For simplicity, binding occurrences of names are annotated with types. We work up to α-equivalence and we assume the convention that all bound names are distinct from each other and from all free names. A process P can be a terminated process $\mathbf{0}$; a parallel composition of processes $P \mid Q$; a replicated process $!P$; an input $x^p?[y\colon M].P$ or an output $x^p![y^p].P$ on a polarised channel x^p with continuation P; a restriction $(\nu x\colon M)\, P$, where name x can be either a

$$x^p\,?\,[y{:}S\,];P \mid x^{\bar{p}}\,!\,[z^q];Q \xrightarrow{x,-} P\left\{{}^{z^q}/_{y}\right\} \mid Q \quad \text{R-Com}$$

$$\frac{P \xrightarrow{\alpha,l} P'}{P \mid Q \xrightarrow{\alpha,l} P' \mid Q}\ \text{R-Par}$$

$$\frac{p \text{ is either } + \text{ or } - \qquad j \in I}{x^p \triangleright \{l_i : P_i\}_{i\in I} \mid x^{\bar{p}} \triangleleft l_j.Q \xrightarrow{x,l_j} P_j \mid Q}\ \text{R-Select}$$

$$\frac{P' \equiv P \quad P \xrightarrow{\alpha,l} Q \quad Q \equiv Q'}{P' \xrightarrow{\alpha,l} Q'}\ \text{R-Cong}$$

$$\frac{P \xrightarrow{\alpha,l} P' \qquad \alpha \neq x \qquad M \text{ not session type}}{(\nu x : M)P \xrightarrow{\alpha,l} (\nu x : M)P'}\ \text{R-New}$$

$$\frac{P \xrightarrow{x,l} P'}{(\nu x : S)P \xrightarrow{\tau,-} (\nu x : tail(S,l))P'}\ \text{R-NewS}$$

Fig. 1. The reduction relation

session channel or a standard channel and finally a branching $x \triangleright \{l_i : P_i\}_{i\in I}$ or selection $x \triangleleft l.P$, modelling respectively, the external and internal choice.

The operational semantics of processes is given in terms of a reduction relation and structural congruence [10] and is presented in Figure 1. The explanation of the fairly standard rules can be found in [10]. The substitution of polarised names for unpolarised variables in rule R-Com is an adaptation of Stoughton's general approach [14], and rule R-NewS uses the auxiliary *tail* function defined as follows,

$$tail(?\,[T];S,\_) = S \quad tail(\&\langle\, l_i{:}\, S_i \,\rangle_{i\in I}, l_j) = S_j$$
$$tail(!\,[T];S,\_) = S \quad tail(\oplus\langle\, l_i{:}\, S_i \,\rangle_{i\in I}, l_j) = S_j \quad tail(\mu X.S, l) = tail(S\left\{{}^{\mu X.S}/_{X}\right\}, l)$$

Reductions are annotated with labels of the form α, l, which indicate the channel name and branching / selection label, if any, being involved in each reduction.

2.3 Type System

In this section we present the typing discipline for the π-calculus with sessions. An environment Γ is a partial function from optionally polarised names to types. We adopt the standard convention that $\Gamma, x^p : M$ is defined if $x^p \notin dom(\Gamma)$. Moreover, the "," operator on environments is defined in a way that respects the polarity of names [10]. An environment Γ is *unlimited* if it contains no session types; *completed* if every session type in Γ is end; and $\mathcal{D}$*-balanced*, for some duality relation $\mathcal{D}$, if whenever $x^+ : S \in \Gamma$ and $x^- : S' \in \Gamma$ then $S\ \mathcal{D}\ S'$. Typing judgements are of the form $\Gamma \vdash P$, meaning "process P is well-typed in the environment Γ". The type system is given in Figure 2. It uses the + operator on environments which combines two type environments without duplication of polarised session channels. The type system differs from the original one in [10] in that subtyping relation is removed from the typing rules and is instead included in the subsumption rule, T-SubsS for session types. Rule T-Nil states that the terminated process **0** is well-typed in any completed environment Γ. Rule T-Par is standard and uses the + operator to combine two environments. Rule T-Rep states that process $!P$ is well-typed if process P is well-typed in the same unlimited environment Γ. Rules T-New and

$$\frac{\Gamma \text{ completed}}{\Gamma \vdash \mathbf{0}}\ \text{T-Nil} \qquad \frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}\ \text{T-Par} \qquad \frac{\Gamma \vdash P \qquad \Gamma \text{ unlimited}}{\Gamma \vdash !P}\ \text{T-Rep}$$

$$\frac{\Gamma, x{:}M \vdash P \qquad M \text{ is not a session type}}{\Gamma \vdash (\nu x{:}M)P}\ \text{T-New} \qquad \frac{\Gamma, x^+{:}S, x^-{:}S' \vdash P \qquad S \perp_{\mathsf{c}} S'}{\Gamma \vdash (\nu x{:}S)P}\ \text{T-NewS}$$

$$\frac{\Gamma, x^p{:}S, y{:}T \vdash P}{\Gamma, x^p{:}?[T];S \vdash x^p?[y{:}T];P}\ \text{T-InS} \qquad \frac{\Gamma, x^p{:}S \vdash P}{(\Gamma, x^p{:}![T];S) + y^q{:}T \vdash x^p![y^q];P}\ \text{T-OutS}$$

$$\frac{\Gamma, x{:}\sharp S, y{:}S \vdash P}{\Gamma, x{:}\sharp S \vdash x?[y{:}S];P}\ \text{T-In} \qquad \frac{\Gamma, x{:}\sharp S \vdash P}{(\Gamma, x{:}\sharp S) + y^q{:}S \vdash x![y^q];P}\ \text{T-Out}$$

$$\frac{\forall i \in I \quad (\Gamma, x^p{:}S_i \vdash P_i)}{\Gamma, x^p{:}\&\langle l_i : S_i\rangle_{i\in I} \vdash x^p \triangleright \{l_i : P_i\}_{i\in I}}\ \text{T-Offer} \qquad \frac{\Gamma, x^p{:}S \vdash P}{\Gamma, x^p{:}\oplus\langle l : S\rangle \vdash x^p \triangleleft l.P}\ \text{T-Choose}$$

$$\frac{\Gamma, x^p : S \vdash P \qquad T \leqslant_{\mathsf{sbt}} S}{\Gamma, x^p : T \vdash P}\ \text{T-SubsS}$$

Fig. 2. Typing rules

T-NewS type the restriction process where the new name x is a shared channel and a session channel, respectively. In the latter, the premise of the rule ensures that the opposite endpoints of the session channel x have dual types according to $\perp_{\mathsf{c}}$. Typing rules for input and output processes are duplicated in order to distinguish between actions on session channels and shared channels. Rule T-InS typechecks an input process that receives messages of a session type T on a session channel x^p of type $?[T];S$. Rule T-In typechecks an input process that receives messages of type S on a shared channel x of type $\sharp\, S$. Rules T-OutS and T-Out typecheck the output process sending messages on session channel x^p and shared channel x, respectively and follow the same line as the typing rules for input processes. Rule T-Offer typechecks branching $x^p \triangleright \{l_i : P_i\}_{i\in I}$ under the assumption that x^p has type $\&\langle l_i : S_i\rangle_{i\in I}$. Rule T-Choose typechecks selection $x^p \triangleleft l.P$ under the assumption that x^p has type $\oplus\langle l : S\rangle$. Notice that the types of the session channel x^p in the previous two rules are exactly what is required for the processes to be well-typed. However, by using subtyping and rule SubsS it is possible to obtain subtypes in breadth and in depth for branch and select types.

The type system enjoys the properties of type preservation and type safety. The proofs can be found in the online version of the paper [1].

Theorem 1 (Type Preservation).

1. *If* $\Gamma \vdash P$ *and* $P \xrightarrow{\tau,-} Q$ *then* $\Gamma \vdash Q$.
2. *If* $\Gamma, x^+{:}S, x^-{:}S' \vdash P$, $S \perp_c S'$, *and* $P \xrightarrow{x,l} Q$ *then* $\Gamma, x^+{:}tail(S,l), x^-{:}tail(S',l) \vdash Q$.
3. *If* $\Gamma, x{:}T \vdash P$ *and* $P \xrightarrow{x,-} Q$ *then* $\Gamma, x{:}T \vdash Q$.

To state the type safety result, we first define safe processes. Given a type environment Γ, we say that a process P is Γ-*safe* whenever:

1. If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x\,?\,[y{:}V]; P_1 \mid x\,![z^q]; P_2 \mid Q)$ then among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x{:}\sharp\, U$ and $z^q{:}W$ with $W \leqslant_{\mathsf{sbt}} U \leqslant_{\mathsf{sbt}} V$.
2. If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x^p\,?\,[y{:}V]; P_1 \mid x^{\overline{p}}\,![z^q]; P_2 \mid Q)$ with $p = {}^+$ or $p = {}^-$ then $x^+, x^- \notin \mathrm{FN}(Q)$ and among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x^p{:}\,?\,[U]; S$ and $x^{\overline{p}}{:}\,!\,[U']; S'$ with $U =_{\mathsf{sbt}} U'$ and $z^q{:}W$ with $W \leqslant_{\mathsf{sbt}} U \leqslant_{\mathsf{sbt}} V$.
3. If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x^p \triangleright \{l_i : P_i\}_{i\in I} \mid x^{\overline{p}} \triangleleft l_j.Q \mid R)$ then $j \in I$ and $x^+, x^- \notin \mathrm{FN}(R)$.

Intuitively a process is Γ-safe whenever it contains no data mismatches between the data expected in the inputs, and the data sent by the outputs. Type safety follows from Proposition 3 and a more general result, Theorem 4, which are given in Section 4.

Corollary 1. *For every process P and $\bot_{\mathsf{c}}$-balanced Γ, if $\Gamma \vdash P$ then P is Γ-safe.*

3 A behavioural theory for session types

Bernardi and Hennessy [2] developed a behavioural theory for session types, in terms of session *contracts*, which gives an extensional meaning to subtyping (Theorem 2 in [2]).

In this section we recall some of their definitions and results, which are needed to give our contribution in the next section. We also consider the traces performed by session contracts, and show that complementation of a contract corresponds to exchanging inputs and outputs in its traces (Lemma 1).

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

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

where we assume the labels $\mathtt{l}_i$ to be pairwise distinct. We use SCts to denote the set of contract terms which are guarded and closed. We refer to these terms as *session contracts*, or just *contracts*.

In order to define the operational meaning of contracts we define the set of visible actions, Act, ranged over by λ, as the union of two sets, namely $\{\, ?\mathtt{l},\ !\mathtt{l} \mid \mathtt{l} \in \mathtt{L} \,\}$, and $\{\, ?(\sigma),\ !(\sigma) \mid \sigma \in \mathsf{SCts} \,\}$. We use Act_τ to denote the set $\mathsf{Act} \cup \{\, \tau \,\}$. We define judgements of the form $\sigma_1 \xrightarrow{\mu} \sigma_2$, where $\mu \in \mathsf{Act}_\tau$ and $\sigma_1, \sigma_2 \in \mathsf{SCts}$, by using standard axioms.

$$\frac{}{\textstyle\sum_{i\in I} ?\mathtt{l}_i.\sigma_i \xrightarrow{?\mathtt{l}_i} \sigma_i} \qquad \frac{}{\mu x.\sigma \xrightarrow{\tau} \sigma\{{}^{\mu x.\sigma}/_{x}\}} \qquad \frac{}{\lambda.\sigma \xrightarrow{\lambda} \sigma}$$

$$\frac{}{\textstyle\bigoplus_{i\in I} !\mathtt{l}_i.\sigma_i \xrightarrow{\tau} !\mathtt{l}_i.\sigma_i}\ |I| > 1 \qquad \frac{}{\textstyle\bigoplus_{i\in I} !\mathtt{l}_i.\sigma_i \xrightarrow{!\mathtt{l}_i} \sigma_i}\ |I| = 1$$

We also have the predicate $\mathsf{1} \xrightarrow{\checkmark}$, which intuitively means that $\mathsf{1}$ is the *satisfied* contract.

In order to define the *peer compliance* relation [2] between contracts ρ, σ, we need to define when these contracts can interact. This is accounted for by judgements of the

form $\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma'$ which are inferred using the following rules.

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

The rules are standard, except for the interaction relation $\bowtie_{\mathcal{B}}$, which in turn is parameterised by a relation $\sigma_1 \ \mathcal{B} \ \sigma_2$ between contracts. Intuitively, $\mathcal{B}$ determines when the contract σ_1 can be accepted when σ_2 is required, and we use $\bowtie_{\mathcal{B}}$ to account for higher-order interactions. The relation $\bowtie_{\mathcal{B}}$ is defined as the union of the sets $\{(?\mathtt{l}, !\mathtt{l}), (!\mathtt{l}, ?\mathtt{l}), \mid \mathtt{l} \in \mathtt{L}\}$, and $\{(?(\sigma_2), !(\sigma_1)), (!(\sigma_1), ?(\sigma_2)) \mid \sigma_1 \ \mathcal{B} \ \sigma_2\}$.

Definition 5 ($\mathcal{B}$-Peer Compliance). *Let $C^{\text{P2P}} : \mathcal{P}(\mathsf{SCts}^2) \times \mathcal{P}(\mathsf{SCts}^2) \longrightarrow \mathcal{P}(\mathsf{SCts}^2)$ be the rule functional defined so that $(\rho, \sigma) \in C^{\text{P2P}}(\mathcal{R}, \mathcal{B})$ whenever (i) if $\rho \parallel \sigma \overset{\tau}{\not\longrightarrow}_{\mathcal{B}}$ then $\rho \xrightarrow{\checkmark}$, and $\sigma \xrightarrow{\checkmark}$; and (ii) if $\rho \parallel \sigma \xrightarrow{\tau}_{\mathcal{B}} \rho' \parallel \sigma'$ then $\rho' \ \mathcal{R} \ \sigma'$.*

Fix a $\mathcal{B}$. If $\mathcal{R} \subseteq C^{\text{P2P}}(\mathcal{R}, \mathcal{B})$, then we say that $\mathcal{R}$ is a $\mathcal{B}$-coinductive peer compliance. *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 $\dashv^{\mathcal{B}}_{\text{P2P}}$.* □

Definition 6 (Peer Subcontract Preorder). *For every $\mathcal{B}$, we write $\sigma_1 \sqsubseteq^{\mathcal{B}} \sigma_2$ whenever $\rho \dashv^{\mathcal{B}}_{\text{P2P}} \sigma_1$ implies $\rho \dashv^{\mathcal{B}}_{\text{P2P}} \sigma_2$ for every ρ. We also let $\mathcal{P}re$ denote the collection of preorders over the set SCts, and let $\mathcal{F}^{\text{P2P}} : \mathcal{P}re \longrightarrow \mathcal{P}re$ be the function defined by $\mathcal{F}^{\text{P2P}}(\mathcal{B}) = \sqsubseteq^{\mathcal{B}}$. We let $\sqsubseteq$ be $\nu X.\mathcal{F}^{\text{P2P}}(X)$, and we refer to $\sqsubseteq$ as the* peer subcontract preorder. □

The full abstraction result depends on a bijection $\mathcal{M}$ between session types and session contracts, which maps END to $\mathsf{1}$, branch types $\&\langle - \rangle$ to external sums $\sum$, choice types $\oplus\langle - \rangle$ to internal sums $\bigoplus$, and leaves the variables and recursion $\mu -.-$ essentially unchanged (see [1, Appendix B]).

Theorem 2 (Full Abstraction [2]). *For every $T, S \in \mathit{SType}$, $S \preccurlyeq_{\mathsf{sbt}} T$ if and only if $\mathcal{M}(S) \sqsubseteq \mathcal{M}(T)$.*

In view of Theorem 2, $S \preccurlyeq_{\mathsf{sbt}} T$ means that the contracts ρ in $\sqsubseteq$-peer compliance with $\mathcal{M}(S)$ are in $\sqsubseteq$-peer compliance also with $\mathcal{M}(T)$. In other words, the contract $\mathcal{M}(T)$ satisfies all the contracts satisfied by $\mathcal{M}(S)$, if the higher-order interactions among contracts are prescribed by $\sqsubseteq$.

The proof of Theorem 2 relies on the complement function $\mathsf{comp} = \mathcal{M} \cdot \mathsf{cplt}$. We will need the property that the contract $\mathsf{comp}(\sigma)$ performs the traces of σ, but with inputs and outputs exchanged. To prove this, let $\overset{s}{\Longrightarrow}$ denote the least relation that satisfies the following conditions, (i) $\sigma \overset{\varepsilon}{\Longrightarrow} \sigma$ for every σ, (ii) $\sigma \overset{\lambda s}{\Longrightarrow} \sigma'$ if $\sigma \xrightarrow{\lambda} \sigma''$ and $\sigma'' \overset{s}{\Longrightarrow} \sigma'$, and (iii) $\sigma \overset{s}{\Longrightarrow} \sigma'$ if $\sigma \xrightarrow{\tau} \sigma''$ and $\sigma'' \overset{s}{\Longrightarrow} \sigma'$. We write $\sigma \overset{s}{\Longrightarrow}$ when $\sigma \overset{s}{\Longrightarrow} \sigma'$ for some contract σ'.

Given a visible action λ, we flip it to $\overline{\lambda}$ by exchanging ! and ?. To exchange inputs and outputs along finite traces we extend $\overline{\cdot}$ inductively to $\mathsf{flip} : \mathsf{Act}^\star \longrightarrow \mathsf{Act}^\star$.

Lemma 1. *For every $\rho \in \mathsf{SCts}$ and $s \in \mathsf{Act}^\star$, $\rho \overset{s}{\Longrightarrow}$ if and only $\mathsf{comp}(\rho) \overset{\mathsf{flip}(s)}{\Longrightarrow}$.*

4 Dualities for session types

In this section we present the set-theoretical comparison of various duality relations defined so far in the literature. These relations are divided into two groups: the first group contains the dualities related to the $\mathcal{B}$-peer compliance obtained by instantiating the parameter $\mathcal{B}$ with the peer preorder $\sqsubseteq$; the second group contains the dualities defined using the inductive duality. We call the first group "peer compliance hierarchy" and the second group "naïve recursion dualities".

4.1 Peer compliance hierarchy

We use the peer subcontract preorder to single out one $\mathcal{B}$-peer compliance relation.

Definition 7 (Peer Compliance). *Let* $\dashv\!\shortmid_{\text{P2P}} = \dashv\!\shortmid^{A}_{\text{P2P}}$ *where* $A = \sqsubseteq$, *and call* $\dashv\!\shortmid_{\text{P2P}}$ *the* peer compliance *relation.* □

The next result gives a syntactic characterisation of peer compliance which also accounts for the mapping between contracts and session types.

Lemma 2 (Syntactic Characterisation of Compliance). *For all session types* S, T, S $\texttt{dual}^{\texttt{sbt}}$ T *if and only if* $\mathcal{M}(S)$ $\dashv\!\shortmid_{\text{P2P}}$ $\mathcal{M}(T)$.

The characterisation given by the previous lemma eases the comparison between the peer compliance and the other dualities for session types. Now we proceed with the formal comparison.

Co-inductive duality is contained in syntactic compliance. This follows mainly from the set inclusion $=_{\text{sbt}} \subseteq \preccurlyeq_{\text{sbt}}$ and from Definition 4.

Corollary 2. $\bot_{\text{c}} \subseteq \texttt{dual}^{\texttt{sbt}}$.

The converse inclusion does not hold. The reason is that $\bot_{\text{c}}$ requires branch or select to have the same set of labels, whereas $\texttt{dual}^{\texttt{sbt}}$ relates also choice types whose sets of labels are in a set inclusion relation.

Example 3. Here we exhibit two types such that S $\texttt{dual}^{\texttt{sbt}}$ T and S $\not\perp_{\text{c}}$ T. Let $S = \&\langle \texttt{moka}\colon \text{END} \rangle$ and $T = \&\langle \texttt{moka}\colon \text{END}, \texttt{tea}\colon \text{END} \rangle$. To prove that S $\texttt{dual}^{\texttt{sbt}}$ T it suffices to observe that $\mathcal{R} \subseteq \mathcal{G}(\mathcal{R}, \preccurlyeq_{\text{sbt}}, \subseteq)$, where $\mathcal{R} = \{ (S, T), (\text{END}, \text{END}) \}$.

The reason why S $\not\perp_{\text{c}}$ T is that by definition $\bot_{\text{c}}$ requires the sets of labels in S and T to be equal and this is false, because $\{\texttt{moka}\} \neq \{\texttt{moka}, \texttt{tea}\}$. □

Next we consider a duality relation based on the *finite* traces performed by session contracts. Let $\mathsf{fTr}(\sigma) = \{ s \in \mathsf{Act}^\star \mid \sigma \stackrel{s}{\Longrightarrow} \}$.

Definition 8 (Trace-Oriented Duality). *Write* $\rho\ \Phi\ \sigma$ *whenever* $\mathsf{fTr}(\rho) = \mathsf{flip}(\mathsf{fTr}(\sigma))$. *If* $\rho\ \Phi\ \sigma$ *we say that* ρ *and* σ have dual finite traces. □

Note that the definition of *multiparty compatibility* in [8] is more general than Definition 8, in that it also accounts for the participants of sessions. However, in the setting of binary session types, our relation Φ and multiparty compatibility coincide up-to minor syntactic annotations.

The co-inductive duality $\perp_{\mathsf{c}}$ — modulo $\mathcal{M}$ — is strictly larger than the trace-based duality. This is true because the traces of a contract ρ and of its unfolding, $\text{UNF}(\rho)$, are the same, as shown in the following.

Lemma 3. *For every session contract ρ, $\mathsf{fTr}(\rho) = \mathsf{fTr}(\text{UNF}(\rho))$.*

Proposition 1. *For every session types S, T, if $\mathcal{M}(S)\ \Phi\ \mathcal{M}(T)$ then $S\ \perp_{\mathsf{c}}\ T$.*

The converse inclusion is false: the coinductive duality $\perp_{\mathsf{c}}$ - modulo $\mathcal{M}$ - is not contained in the relation Φ. As we show in the next example, the reason is that trace equality does not compare contracts modulo unfolding.

Example 4. We show two types T_1 and T_2 such that $T_1\ \perp_{\mathsf{c}}\ T_2$ and $\mathcal{M}(T_1)\ \Phi\ \mathcal{M}(T_2)$ is false. Let S_1 be $?[\,\text{END}\,].S_2$ where $S_2 = \mu X.?[\,\text{END}\,].X$, and let $T_1 = ![\,S_1\,].\text{END}$, and $T_2 = ?[\,T_2\,].\text{END}$.

First we prove that $T_1\ \perp_{\mathsf{c}}\ T_2$. Definition 4 requires us to exhibit a suitable co-inductive duality, namely $\mathcal{R} = \{(T_1, T_2), (\text{END}, \text{END})\}$, and a suitable co-inductive subtyping, namely $\mathcal{S} \cup \mathcal{S}^{-1}$, where $\mathcal{S} = \{(S_1, S_2), (S_2, S_2), (\text{END}, \text{END})\}$.

Now we show that $\mathcal{M}(T_1)\ \Phi\ \mathcal{M}(T_2)$ is false. It suffices to exhibit a trace $s \in \mathsf{fTr}(\mathcal{M}(T_1))$ such that $s \notin \mathsf{flip}(\mathsf{fTr}(\mathcal{M}(T_2)))$. Notice that $\mathcal{M}(T_1) \overset{!(\sigma_1)}{\Longrightarrow}$ with $\sigma_1 = \mathcal{M}(S_1)$, so $!(\sigma_1) \in \mathsf{fTr}(\mathcal{M}(T_1))$, and that $\mathsf{fTr}(\mathcal{M}(T_2)) = \{\varepsilon, ?(\sigma_2)\}$ and $\sigma_2 = \mathcal{M}(S_2) \neq \mathcal{M}(S_1) = \sigma_1$, so $?(\sigma_1) \notin \mathsf{flip}(\mathsf{fTr}(\mathcal{M}(T_2)))$. □

The trace-oriented duality is strictly larger than the complement function. This is true for two reasons, one is that the complementary contracts have flipped traces (see Lemma 1); the other is that the trace-oriented duality relates contracts with their unfoldings, whereas the complement function does not (see Example 5).

Corollary 3. *For all $\rho \in \mathsf{SCts}$, $\rho\ \Phi\ \mathsf{comp}(\rho)$.*

Example 5. We exhibit two contracts ρ and σ such that $\rho\ \Phi\ \sigma$ and $(\rho, \sigma) \notin \mathsf{comp}$. Let $\rho = \mu x.?(1).x$ and $\sigma = !(1).\mu x.!(1).x$. Plainly ρ and σ have the same finite traces up-to an application of $\mathsf{flip}(\cdot)$, thus $\mu x.?(1).x\ \Phi ?(\text{END}).x.\mu x.?(1).x$. However, $\mathsf{comp}(\mu x.?(1).x) = \mu x.!(1).x \neq !(1).\mu x.!(1).x$, so $(\rho, \sigma) \notin \mathsf{comp}$. □

Finally, we relate complement, subtyping and syntactic compliance.

Proposition 2. $\mathtt{dual}^{\mathsf{sbt}} = \mathsf{cplt} \cdot \leqslant_{\mathsf{sbt}} = \leqslant_{\mathsf{sbt}} \cdot \mathsf{cplt}$.

Figure 3 summarises the results of this section, where the symbols $=_{\mathcal{M}}$, and $\subseteq_{\mathcal{M}}$ stand for set equality and set inclusion up-to encoding via $\mathcal{M}$, respectively.

4.2 Naïve recursion dualities

The duality functions $\overline{(\cdot)}$ and cplt do not relate types modulo unfolding, for instance in general $\mathsf{cplt}(T) \neq \mathsf{cplt}(\text{UNF}(T))$. To overcome this limitation, Vallecillo *et al.* [16] defined a *compatibility* relation by composing inductive duality with subtyping. Gay and Vasconcelos [11] took a similar approach, but defined compatibility differently.

$$\mathsf{comp} \subsetneq \Phi \subsetneq_{\mathcal{M}} \bot_{\mathsf{c}} \subsetneq \ \mathtt{dual}^{\mathtt{sbt}} \ =_{\mathcal{M}} \dashv\!\!\!|_{\mathrm{P2P}}$$
$$=$$
$$\mathsf{cplt} \cdot \ \preccurlyeq_{\mathsf{sbt}} = \ \preccurlyeq_{\mathsf{sbt}} \cdot \mathsf{cplt}$$

Fig. 3. Hierarchy of duality relations

Definition 9 (Compatibility [16, 11]).
Let $\bowtie = \{(T,\, S) \mid T \preccurlyeq_{\mathsf{sbt}} \overline{S}\}$*, and* $\asymp = \{(T,\, S) \mid \overline{T} \preccurlyeq_{\mathsf{sbt}} S\}$. □

As far as we know, the relations $\bowtie$ and $\asymp$ were intended to coincide and to be symmetric. However, by considering the type $\mu X.\, ![\, X\,].\textsc{end}$ and its inductive dual, we find that $\bowtie$ and $\asymp$ are different, and that neither relation is symmetric:

$$\mu X.\, ![\, X\,].\textsc{end} \asymp\ ?[\, \mu X.\, ?[\, X\,].\textsc{end}\,].\textsc{end} \qquad ?[\, \mu X.\, ?[\, X\,].\textsc{end}\,].\textsc{end} \not\asymp \mu X.\, ![\, X\,].\textsc{end}$$
$$\mu X.\, ![\, X\,].\textsc{end} \not\bowtie\ ?[\, \mu X.\, ?[\, X\,].\textsc{end}\,].\textsc{end} \qquad ?[\, \mu X.\, ?[\, X\,].\textsc{end}\,].\textsc{end} \bowtie \mu X.\, ![\, X\,].\textsc{end}$$

The lack of symmetry contradicts Proposition 3.3 in [16] and Proposition 3 in [11].

Proposition 2 ensures that if one defines compatibility in terms of complement instead of in terms of inductive duality then the resulting relations are equivalent, and coincide with the relation $\mathtt{dual}^{\mathtt{sbt}}$, which is symmetric. Lemma 2 and the inclusion $\dashv\!\!\!|_{\mathrm{P2P}} \cdot \sqsupseteq \subseteq \dashv\!\!\!|_{\mathrm{P2P}}$ suggest that the peer compliance $\dashv\!\!\!|_{\mathrm{P2P}}$ captures the intuition behind the compatibility relations.

4.3 The typing relation as a function of duality

In order to show the impact that different dualities have on the typing relation $\vdash$, we parameterise its definition by a binary relation $\mathcal{D}$ on session types. Let $\vdash_{\mathcal{D}}$ be the relation defined by the rules in Fig. 2, where rule T-NewS is replaced by the following rule,

$$\frac{\Gamma, x^{+} : S, x^{-} : S' \vdash_{\mathcal{D}} P \qquad S \ \mathcal{D}\ S'}{\Gamma \vdash_{\mathcal{D}} (\nu x : S)P}\ \text{T-NewS'}$$

The relation $\vdash_{\mathcal{D}}$ is a function of $\mathcal{D}$ and it is monotone.

Proposition 3. *For every* $\mathcal{D} \subseteq \mathcal{D}' \subseteq \mathit{SType}^2$, $\vdash_{\mathcal{D}} \subseteq \vdash_{\mathcal{D}'}$.

Considering the hierarchy in Figure 3, from $\mathsf{comp} \subsetneq_{\mathcal{M}} \bot_{\mathsf{c}}$ and the definitions of cplt and $\mathcal{M}$ we have $\mathsf{cplt} \subsetneq \bot_{\mathsf{c}}$; also $\bot_{\mathsf{c}} \subsetneq \mathtt{dual}^{\mathtt{sbt}}$. Proposition 3 then yields:

Corollary 4. *For every* Γ *and* P, $\Gamma \vdash_{\mathsf{cplt}} P$ *implies* $\Gamma \vdash_{\bot_{\mathsf{c}}} P$, *and* $\Gamma \vdash_{\bot_{\mathsf{c}}} P$ *implies* $\Gamma \vdash_{\mathtt{dual}^{\mathtt{sbt}}} P$.

The difference between cplt and $\bot_{\mathsf{c}}$ is that $\bot_{\mathsf{c}}$ includes arbitrary unfolding, and the difference between $\bot_{\mathsf{c}}$ and $\mathtt{dual}^{\mathtt{sbt}}$ is that $\mathtt{dual}^{\mathtt{sbt}}$ includes subtyping. Since the rule T-SubsS allows the use of subtyping relation in the derivation trees, also the converse of Corollary 4 is true:

Theorem 3. *$\Gamma \vdash_{\mathsf{cplt}} P$ if and only if $\Gamma \vdash_{\perp_c} P$ if and only if $\Gamma \vdash_{\mathtt{dual}^{\mathtt{sbt}}} P$.*

Example 6. Here we show that Theorem 3 is false if in T-SUBSS we replace $\leqslant_{\mathsf{sbt}}$ with the equivalence $=_{\mathsf{sbt}}$. Let $T = \oplus\langle\, a : \text{END} \,\rangle$, $S = \&\langle\, a : \text{END}, b : \text{END} \,\rangle$, $P = (\nu x{:}S)(x^{+} \triangleright \{a : \mathbf{0}, b : \mathbf{0}\} \mid x^{-} \triangleleft a.\mathbf{0})$, and let the symbol $\vdash'_{\mathcal{D}}$ denote the type relation given by the rules in Figure 2, where T-NEWS' is used in place of T-NEWS. The derivation of $\vdash'_{\mathtt{dual}^{\mathtt{sbt}}} P$ exists, because $S \ \mathtt{dual}^{\mathtt{sbt}} \ T$, while a derivation of $\vdash'_{\perp_c} P$ does not exist, because $S \perp_c T$ is false. □

Type safety is true also for the dualities cplt and $\mathtt{dual}^{\mathtt{sbt}}$.

Theorem 4. *For every process P, if $\Gamma \vdash_{\mathtt{dual}^{\mathtt{sbt}}} P$ and Γ is $\mathtt{dual}^{\mathtt{sbt}}$-balanced, then P is Γ-safe*

Two immediate consequences are Corollary 1, previously stated, and the analogous result for cplt. The proof that cplt is a safe duality is essential, and was not given in [2].

5 Related Work

Session types: The concept of a session type originated in the work of Honda *et al.* [15, 12]. There is now a substantial literature, which has been surveyed by Dezani and de'Liguoro [9]. Subtyping for recursive session types was proposed in [10], and full abstraction for that subtyping was achieved in [2]. Another semantic theory of session types is defined in [5], and it is independent of the theory of [10].

Subtyping: Recent work by Chen *et al.* [6] has considered *preciseness* of subtyping, meaning the combination of soundness and completeness. A subtyping relation is complete if it is the largest safe subtyping relation for a given type system. Given the connections between subtyping and duality (e.g. Proposition 2), the results about preciseness are likely to be relevant for finding the largest safe duality. On the other hand, if one removes subtyping from the type system, then the choice of duality relation becomes relevant and impacts the resulting typing relation. We leave the investigation of the relation between subtyping and duality for future work.

Complement and inductive duality: The only difference between the complement function and the inductive duality lies in the treatment of recursive terms. Although apparently small, this difference leads to noticeable differences in the typing relations (see [2, Section 5]). In turn, the different treatment of recursion boils down to the use of the substitution $\lfloor \cdot / . \rfloor$. This substitution was first proposed in [3], where it is called *inner substitution*. The reason why [3, 2] use this substitution is to inductively construct the duals of types like $\mu X.\,![\,X\,].\text{END}$. This is borne out in [3, Proposition 3.1 (2)] and [2, Theorem 5.10]. The duality of [3] is syntactically defined, whereas the $\mathcal{B}$-mutual compliances are operationally defined. As a consequence the proof of Proposition 3.1 (2) is simpler than the proof of [2, Theorem 5.10]. Also, while [3] uses the fact that complement and unfold commute, the authors of [2] prove this result.

Further dualities: One extensional duality which we have not considered in this comparison is the one of [5, Definition 2.5]. The reason for this omission is that the framework of that paper is significantly different from ours. However, we leave the comparison with the dualities we studied here and the one of [5] for future work.

Two other approaches to defining duality emerged from the connection between session types and propositions [4, 17]. In the intuitionistic setting of Caires and Pfenning [4], duality boils down to the cut-rule, according to which the dual of requiring a behaviour A (on the left of a judgement) is an offer of A (on the right of a judgment). This implicit duality is in contrast with the situation in the classical setting studied by Wadler [17], where duality of session types amounts to duality on propositions: every proposition A has an inductively defined explicit dual $A^{\perp}$.

Finally, an interesting aspect of duality on session types is its relation to the opposite input / output channel capabilities in the standard π-calculus. In [7] the authors study an encoding of binary session types into linear channel π-types. Due to the *continuation-passing style* of the encoding, the duality on session types reduces to merely opposite input / output linear channel capabilities, but only in the outermost level of the type, with the carried types being (syntactically) equal.

6 Conclusion and Future Work

The duality relation is an essential and distinctive feature of session type systems. However, the questions of how to define duality, and of the effect of duality on the typing relation, have not received careful attention; different authors have used different definitions for various reasons of convenience without considering the consequences. Particular care is needed in the presence of recursive types, and the informal convention of working "up to unfolding" is problematic. In the present paper, we have begun a systematic study of duality relations, and we have made the following contributions.

First, we have emphasised the problems of treating session types "up to unfolding", and we have carefully defined a session type system without this assumption.

Second, we have explicitly considered a range of duality relations and established their relationships. We have included several existing relations defined on the syntax of session types, and some new relations derived from a semantic model of session types.

Third, we have considered the duality relation as a parameter of the session type system, and studied the typing relations resulting from different choices of duality relation. The result is that the type system is robust, in the sense that the choice of duality relation among the syntax-based relations does not matter.

Future work. The model of session types developed in [2] does not account for standard channel types. This is a severe limitation, because in settings based on that model, standard channels cannot be sent over session channels. This limitation applies to the present paper. We plan to investigate how to extend the model of [2] to channel types.

If a duality $\mathbb{D}$ allows unsafe processes to be typed, then there is no point in using it, so the obvious basis for judging a duality relation $\mathbb{D}$ is the definition of safe process. In turn, once we fix the set of safe processes $\mathcal{S}$, an obvious question is: what is the greatest duality $\mathbb{D}$ such that $\vdash_{\mathbb{D}} P$ implies $P \in \mathcal{S}$? To the best of our knowledge this problem has never been investigated. Once such a greatest $\mathbb{D}$ is found, a similar problem is to find the *smallest* duality d such that $\vdash_d = \vdash_{\mathbb{D}}$. An answer to this problem might have worthwhile practical consequences: for example, an algorithm to decide d may be more efficient than one to decide $\mathbb{D}$. Identifying $\mathbb{D}$ and d would also answer the open question of what should be the *canonical* definition of duality for session types.

Acknowledgements. Bernardi was supported by the Portuguese FCT project PTDC/EIA-CCO/122547/2010. Dardha, Gay and Kouzapas are supported by the UK EPSRC project *From Data Types to Session Types: A Basis for Concurrency and Distribution (ABCD)* (EP/K034413/1). Kouzapas is also supported by an EPSRC Doctoral Prize Fellowship. The research reported in this paper was made possible by a Short-Term Scientific Mission grant to Bernardi from COST Action IC1201: Behavioural Types for Reliable Large-Scale Software Systems (BETTY).

References

1. G. Bernardi, O. Dardha, S. J. Gay, and D. Kouzapas. On duality relations for session types. Online version, 2014. Available at `http://www.dcs.gla.ac.uk/~ornela/my_papers/BDGK14-Extended.pdf`.
2. G. Bernardi and M. Hennessy. Using higher-order contracts to model session types. *CoRR*, abs/1310.6176, 2013.
3. V. Bono and L. Padovani. Typing copyless message passing. *Logical Methods in Computer Science*, 8(1), 2012.
4. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, pages 222–236, 2010.
5. G. Castagna, M. Dezani-Ciancaglini, E. Giachino, and L. Padovani. Foundations of session types. In A. Porto and F. J. López-Fraguas, editors, *PPDP*, pages 219–230. ACM, 2009.
6. T. Chen, M. Dezani-Ciancaglini, and N. Yoshida. On the preciseness of subtyping in session types. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming (PPDP)*, 2014.
7. O. Dardha, E. Giachino, and D. Sangiorgi. Session types revisited. In *PPDP*, pages 139–150. ACM, 2012.
8. P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP, Springer LNCS 7966*, pages 174–186, 2013.
9. M. Dezani-Ciancaglini and U. de'Liguoro. Sessions and session types: An overview. In C. Laneve and J. Su, editors, *WS-FM, Spring LNCS 6194*, volume 6194 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2009.
10. S. J. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005.
11. S. J. Gay and V. T. Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010.
12. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP, Springer LNCS 1381*, pages 122–138, 1998.
13. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
14. A. Stoughton. Substitution revisited. *Theor. Comput. Sci.*, 59:317–325, 1988.
15. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE, Springer LNCS 817*, pages 398–413, 1994.
16. A. Vallecillo, V. T. Vasconcelos, and A. Ravara. Typing the behavior of software components using session types. *Fundam. Inform.*, 73(4):583–598, 2006.
17. P. Wadler. Propositions as sessions. In *ICFP*, pages 273–286, 2012.