Duality of Session Types: The Final Cut

Simon J. Gay
School of Computing Science
University of Glasgow, UK
Simon.Gay@
glasgow.ac.uk

Peter Thiemann
Institut für Informatik
University of Freiburg, Germany
thiemann@
informatik.uni-freiburg.de

Vasco T. Vasconcelos
Faculdade de Ciências
University of Lisbon, Portugal
vmvasconcelos@
ciencias.ulisboa.pt

Duality is a central concept in the theory of session types. Since a flaw was found in the original definition of duality for recursive types, several other definitions have been published. As their connection is not obvious, we compare the competing definitions, discuss tradeoffs, and prove some equivalences. Some of the results are mechanized in Agda.

1 Introduction

Duality is a central concept in the theory of session types. If S is a session type describing a two-party interaction from the viewpoint of one party, then $\overline{S}$ describes the interaction from the viewpoint of the other party. For example, $S = \mu X.?\mathsf{int}.X$ describes indefinitely receiving integers, and its dual $\overline{S} = \mu X.!\mathsf{int}.X$ describes indefinitely sending integers. If the users of the two endpoints of a channel follow types S and $\overline{S}$, respectively, then correct communication takes place.

The original papers on session types [7, 8, 11] define the dual $\overline{S}$ of a session type S by structural induction on S:

$$\overline{\mathsf{end}} = \mathsf{end} \qquad \overline{!T.S} = ?T.\overline{S} \qquad \overline{?T.S} = !T.\overline{S}$$

Recursion is only introduced in the last paper in the series, [8], where recursive session types are handled via the following rules.

$$\overline{X} = X \qquad \overline{\mu X.S} = \mu X.\overline{S}$$

With this definition, indeed we have $\overline{\mu X.?\mathsf{int}.X} = \mu X.!\mathsf{int}.X$, given that duality exchanges input and output. Gay & Hole [5, 6] define a more general duality *relation* $\perp$ so that, for example, $\mu X.?\mathsf{int}.X \perp \mu X.!\mathsf{int}.!\mathsf{int}.X$. The definition is coinductive. This relation gives greater flexibility in typing derivations and follows the idea that duality is a behavioural relation on automata. The relationship between the duality function and the duality relation is intended to be that $\overline{S} \perp S$ for every session type S.

Bernardi & Hennessy [3, 4] show that the duality function $\overline{(\cdot)}$ violates the duality relation for recursive session types when the recursion variable can occur as the type of a message, as in $S = \mu X.?X.X$. In this example, we have $\overline{S} = \mu X.!X.X$. Noting that an occurrence of X stands for the whole μ type, the type of the message in S is S but the type of the message in $\overline{S}$ is $\overline{S}$. In other words, we have dual types in which the type of the message being sent is not the same as the type of the message being received, which violates soundness of any type system that uses this definition of duality. We refer to $\overline{(\cdot)}$ as *naive duality* because it initially seems reasonable but is not correct in all situations.

One way to solve this problem is to require that recursion variables only occur in tail position in a session type, such as X in $?T.X$. As far as we know, almost all papers that use naive duality can be fixed by restricting to tail recursion, because their examples and applications are all tail recursive. One exception is a paper by Vasconcelos [12], which has an interesting application of the type $\mu X.!X.X$ to encode replication, but that could be easily solved with a tail recursive type at the expense of creating an extra channel. Bernardi & Hennessy give examples of pi-calculus processes that can only be typed by using non-tail-recursive session types, but they are specially crafted for the purpose. Nevertheless, it is more satisfactory to have a duality function that works for all session types.

Bernardi & Hennessy [4] give an alternative, correct duality function and justify it with respect to their model of session types which is based on a theory of contracts. The key idea is that a session type can be converted into an equivalent type in which all message types are closed. In Section 3 we present their definition and a variation of it, and reformulate their correctness result in a standard model of recursive types.

Bernardi, Dardha, Gay & Kouzapas [2] discuss several definitions of duality, focusing on the fact that there can be different sound definitions which give rise to different typing relations. One of their definitions is that of Bernardi & Hennessy [4]. They point out that some results claimed by Gay & Hole [6] are false for non-tail-recursive types.

Lindley & Morris [9] give another definition of the duality function. It maps a type variable X in tail position to a *negative type variable* $\overline{X}$, but in a message position it remains as X. As well as being a technical convenience, negative variables allow interesting types such as $\mu X.!\overline{X}.X$. Lindley & Morris justify their definition on general type-theoretic grounds, but do not directly prove its correctness with respect to the duality relation. In Section 4 we do so, as well as giving an equivalent and arguably simpler variation, and another variation that turns out to be equivalent to the Bernardi-Hennessy definition.

As well as proving the results mentioned above on paper, we have begun work on mechanising them in Agda. We summarise the mechanisation in Section 5.

2 Basic Definitions about Session Types

We work formally with a subset of session types, consisting of input and output (no branch or select), and int as a representative data type. All definitions and proofs can be straightforwardly extended to cover branch and select; reasoning about equivalence and duality of session types is not affected by the details of data types.

Definition 1 (Types and Session Types) *Let X,Y,Z range over a denumerable set of* type variables*. Types (T,U) and session types (R,S) are defined by*

$$T,U ::= \mathsf{int} \mid S \qquad\qquad R,S ::= \mathsf{end} \mid ?T.S \mid !T.S \mid X \mid \mu X.S$$

Session types must be contractive*, meaning that they must not contain sub-expressions of the form $\mu X.\mu X_1.\ldots\mu X_n.X$ for $n \geq 0$. The expression $\mu X.S$ binds type variable X with scope S. The set $\mathsf{fv}(T)$ of* free type variables *in a type T is defined as usual, and so is α*-congruence*. The set of closed session types is denoted by* SType *and the set of closed types is denoted by* Type*, so that* $\mathsf{Type} = \mathsf{SType} \cup \{\mathsf{int}\}$*. We identify types that are α-congruent and follow the Barendregt convention on variables [1].*

In the sequel, we use term *type* for any contractive type generated by the grammar for T. When we mean a closed type, we shall speak of $T \in \mathsf{Type}$. The same reasoning applies to session types, where the term *session type* denotes a contractive type generated by the grammar for S, and $S \in \mathsf{SType}$ denotes a closed session type.

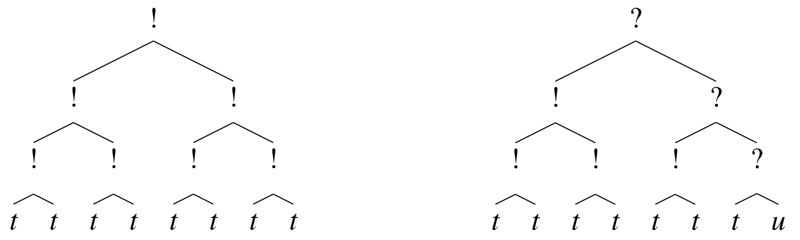
Definition 2 (Substitution) *The result of substituting type U for the free occurrences of variable X in type T—notation $T[U/X]$—is defined inductively as follows, where $Y \neq X$.*

$$\begin{array}{llll}
\mathsf{int}[U/X] = \mathsf{int} & (?S.T)[U/X] = ?S[U/X].T[U/X] & X[U/X] = U & (\mu X.S)[U/X] = \mu X.S \\
\mathsf{end}[U/X] = \mathsf{end} & (!S.T)[U/X] = !S[U/X].T[U/X] & Y[U/X] = Y & (\mu Y.S)[U/X] = \mu Y.S[U/X]
\end{array}$$

Closed session types are interpreted as regular trees in the standard way presented by Pierce [10, Chapter 21]. A regular tree is a (possibly infinite) tree with a finite number of distinct subtrees.

Definition 3 (Types as Trees) *Types are represented by regular trees whose nodes are taken from the set* $\{\mathsf{int}, \mathsf{end}, !, ?\}$, int *and* end *have no descendants,* $!$ *and* $?$ *have two descendants, and* int *can only occur as root or at the immediate left of* $!$ *or* $?$. *We write* $\mathsf{treeof}(T)$ *for the tree representation of T.*

Example 4 *Let S be the session type $\mu X.!X.X$. The regular tree $t = \mathsf{treeof}(S)$ can be depicted as below left. The tree u such that $t \asymp u$ can be depicted as below right.*



Equivalence of session types is equality of trees. We give a coinductive syntactic characterisation of equivalence.

Definition 5 (Syntactic Equivalence of Types) *If $\mathscr{E}$ is a relation on* Type *then $F_{\approx}(\mathscr{E})$ is the relation on* Type *defined by:*

$$\begin{aligned}
F_{\approx}(\mathscr{E}) = {} & \{(\mathsf{end}, \mathsf{end})\} \\
\cup {} & \{(\mathsf{int}, \mathsf{int})\} \\
\cup {} & \{(?T_1.S_1, ?T_2.S_2) \mid (T_1, T_2), (S_1, S_2) \in \mathscr{E}\} \\
\cup {} & \{(!T_1.S_1, !T_2.S_2) \mid (T_1, T_2), (S_1, S_2) \in \mathscr{E}\} \\
\cup {} & \{(S_1, \mu X.S_2) \mid (S_1, S_2[\mu X.S_2/X]) \in \mathscr{E}\} \\
\cup {} & \{(\mu X.S_1, S_2) \mid (S_1[\mu X.S_1/X], S_2) \in \mathscr{E} \text{ and } S_2 \neq \mu Y.S_3\}
\end{aligned}$$

A relation $\mathscr{E}$ on Type *is a* type bisimulation *if $\mathscr{E} \subseteq F_{\approx}(\mathscr{E})$. Syntactic equivalence of types, $\approx$, is the largest type bisimulation.*

Proposition 6 (Type equivalence is tree equality [10]) *Let $T, U \in$* Type. *Then $T \approx U$ if and only if* $\mathsf{treeof}(T) = \mathsf{treeof}(U)$.

The duality relation is defined on regular trees.

Definition 7 (Duality on Trees) *Two trees, s and t, are related by duality—notation $s \asymp t$—if they have the same structure and, for each pair of corresponding nodes, if the nodes are* in the right spine of the tree *they are related as below, otherwise they are the same.*

$$\mathsf{int} \leftrightarrow \mathsf{int} \qquad\qquad \mathsf{end} \leftrightarrow \mathsf{end} \qquad\qquad ? \leftrightarrow ! \qquad\qquad ! \leftrightarrow ?$$

Because $\asymp$ is bijective and every tree is related to some other (unique) tree, we can also regard it as a self-inverse function, which we denote by $\mathsf{dual}(\cdot)$.

Proceeding as for type equivalence, we now give a coinductive syntactic characterisation of the duality relation, restricting attention to session types because int can only occur in message positions, where duality is never applied. This principle is applied to all of our syntactic definitions of duality.

Definition 8 (Syntactic Duality of Session Types) *If $\mathscr{D}$ is a relation on* SType *then $F_{\perp}(\mathscr{D})$ is the relation on* SType *defined by:*

$$\begin{aligned} F_{\perp}(\mathscr{D}) = \{&(\mathsf{end},\mathsf{end})\} \\ \cup\{&(?T_1.S_1, !T_2.S_2) \mid T_1 \approx T_2 \textit{ and } (S_1,S_2) \in \mathscr{D}\} \\ \cup\{&(!T_1.S_1, ?T_2.S_2) \mid T_1 \approx T_2 \textit{ and } (S_1,S_2) \in \mathscr{D}\} \\ \cup\{&(S_1, \mu X.S_2) \mid (S_1, S_2[\mu X.S_2/X]) \in \mathscr{D}\} \\ \cup\{&(\mu X.S_1, S_2) \mid (S_1[\mu X.S_1/X], S_2) \in \mathscr{D} \textit{ and } S_2 \neq \mu Y.S_3\} \end{aligned}$$

A relation $\mathscr{D}$ on SType *is a* session duality *if $\mathscr{D} \subseteq F_{\perp}(\mathscr{D})$. Duality of session types, $\perp$, is the largest session duality.*

Proposition 9 (Type Duality Is Tree Duality) *Let $R,S \in$* SType. *Then $R \perp S$ if and only if* $\mathsf{treeof}(R) \asymp \mathsf{treeof}(S)$.

Proof Similar to that of Proposition 6. □

This section introduces duality as a relation on session types. It turns out that, given a session type S, one can construct a session type S' such that $\mathsf{treeof}(S) \asymp \mathsf{treeof}(S')$, or equivalently $S \perp S'$. The next two sections show two different approaches to the problem, both starting from session types in syntactic form (Definition 1).

3 Duality à la Bernardi-Hennessy

Bernardi and Hennessy [4] observe that the problem of building a dual session type with naive duality (as explained in Section 1) is caused by free variables in message types. They give a method for constructing a dual type for an arbitrary session type S:

1. Convert S into an equivalent type S' in which every message type is closed. This step is called *message closure* (Definition 14, later).
2. Apply naive duality to S' (Definition 10, below).

In this section we present the details of this approach. First we gather the definition of naive duality from Section 1.

Definition 10 (Naive Duality Function) *The naive duality function on session types is inductively defined as follows.*

$$\overline{?T.S} = !T.\overline{S} \qquad \overline{!T.S} = ?T.\overline{S} \qquad \overline{\mathsf{end}} = \mathsf{end} \qquad \overline{X} = X \qquad \overline{\mu X.S} = \mu X.\overline{S}$$

We use the term *tail recursive* for session types in which all message types are closed. It turns out that these are *not* types with variables in tail positions only. A counterexample is $\mu X.!(?\mathsf{int}.X).\mathsf{end}$ where X occurs in tail position, but there is a message type that is not closed, namely $?\mathsf{int}.X$. To define tail recursive types we introduce a type formation system that essentially keeps track of the free variables in processes, in such a way that types such as the above are deemed ill-formed.

Definition 11 (Tail Recursive Types) *Let $\mathscr{X}$ be a set of type variables. The set of* tail recursive types *over $\mathscr{X}$, notation $\mathscr{X} \vdash T$, is defined inductively as follows.*

$$\frac{}{\mathscr{X} \vdash \mathsf{int}} \qquad \frac{}{\mathscr{X} \vdash \mathsf{end}} \qquad \frac{\emptyset \vdash S \quad \mathscr{X} \vdash T}{\mathscr{X} \vdash ?S.T} \qquad \frac{\emptyset \vdash S \quad \mathscr{X} \vdash T}{\mathscr{X} \vdash !S.T} \qquad \frac{X \in \mathscr{X}}{\mathscr{X} \vdash X} \qquad \frac{\mathscr{X}, X \vdash T}{\mathscr{X} \vdash \mu X.T}$$

The set of tail recursive types *is the set of types T such that $\emptyset \vdash T$.*

We can easily see that, if $\mathscr{X} \vdash \mu X.!S.T$, then X does not occur free in S. In particular the type $\mu X.!(?\mathsf{int}.X).\mathsf{end}$ identified above is not tail recursive.

Gay & Hole [6] claim to prove that for all $S \in \mathsf{SType}$, $\overline{S} \perp S$. They use a slightly different definition of $\perp$ in which types are completely unfolded before analysing their structure. Unfolding means repeatedly transforming top-level $\mu X.S$ to $S[\mu X.S/X]$ until a non-μ type is exposed. However, the proof contains the claim that if the unfolding of S is $?T.S'$ then the unfolding of $\overline{S}$ is $!T.\overline{S'}$, which is not true if T contains type variables. Their proof does, however, show the following result.

Proposition 12 (Soundness of Naive Duality for Tail Recursive Types [6]) *If S is a tail recursive type, then $S \perp \overline{S}$.*

This supports the Bernardi-Hennessy approach, because it shows that if a session type can be converted to an equivalent type that is tail recursive, then it is sufficient to apply naive duality to the tail recursive type.

Message closure builds a tail recursive session type by collecting substitutions $[\mu X.S/X]$ for each $\mu X.S$ type encountered and applying the accumulated substitution to messages. To define message closure we need the notion of a sequence of substitutions.

Definition 13 (Sequence of Substitutions) *A* sequence of substitutions *is given by the following grammar:*

$$\sigma ::= \varepsilon \mid [S/X];\sigma$$

The application of a sequence of substitutions σ *to a type T—notation $T\sigma$—is defined as $T\varepsilon = T$ and $T([S/X];\sigma) = (T[S/X])\sigma$. A sequence of substitutions σ is* closing *for T if $\mathsf{fv}(T\sigma) = \varepsilon$.*

Definition 14 (Message Closure [4]) *For any type T and sequence of substitutions σ closing for T, the type $\mathsf{mclo}(T,\sigma)$ is defined inductively by the following rules.*

$$\begin{aligned} \mathsf{mclo}(\mathsf{end},\sigma) &= \mathsf{end} & \mathsf{mclo}(X,\sigma) &= X \\ \mathsf{mclo}(!T.S,\sigma) &= !(T\sigma).\mathsf{mclo}(S,\sigma) & \mathsf{mclo}(\mu X.S,\sigma) &= \mu X.\mathsf{mclo}(S,[(\mu X.S)/X];\sigma) \\ \mathsf{mclo}(?T.S,\sigma) &= ?(T\sigma).\mathsf{mclo}(S,\sigma) \end{aligned}$$

Define $\mathsf{mclo}(S)$ as $\mathsf{mclo}(S,\varepsilon)$.

Bernardi and Hennessy prove that taking the naive dual of the message closure of a type is sound with respect to a notion of compatibility based on a labelled transition system for session types. We will prove soundness with respect to regular trees. First, however, we show that if $S \in \mathsf{SType}$ then $\mathsf{mclo}(S)$ is tail recursive.

The next two lemmas are easily proved by induction.

Lemma 15 *If $\mathscr{X} \vdash T$, then $\mathsf{fv}(T) \subseteq \mathscr{X}$.*

Lemma 16 (Strengthening) *If $\mathscr{X}, X \vdash T$ and $X \notin \mathsf{fv}(T)$, then $\mathscr{X} \vdash T$.*

Combining them, we can identify exactly the type variables that occur free in message positions.

Corollary 17 *If $\mathscr{X} \vdash T$, then $\mathsf{fv}(T) \vdash T$.*

Proof From Lemma 15 we know that $\mathsf{fv}(T) \subseteq \mathscr{X}$. Use Strengthening (Lemma 16) repeatedly to remove from $\mathscr{X}$ type variables not in $\mathsf{fv}(T)$. □

Finally, we reason about $\mathsf{mclo}(T,\sigma)$.

Lemma 18 *If σ is a closing substitution for T, then* $\mathsf{dom}(\sigma) \vdash \mathsf{mclo}(T,\sigma)$.

Proof Straightforward induction on the definition of $\mathsf{mclo}(T,\sigma)$. □

Corollary 19 *If T is closed, then* $\mathsf{mclo}(T)$ *is tail recursive.*

Proof If T is closed, then ε is a closing substitution for T. Lemma 18 ensures that $\emptyset \vdash \mathsf{mclo}(T,\varepsilon)$, hence $\emptyset \vdash \mathsf{mclo}(T)$ by definition. □

Example 20 *The Bernardi-Hennessy approach to duality applied to our running example $S = \mu X.!X.X$.*

$$\begin{aligned}\overline{\mathsf{mclo}(S)} = \overline{\mathsf{mclo}(S,\varepsilon)} &= \overline{\mu X.\mathsf{mclo}(!X.X,[S/X])}\\ &= \overline{\mu X.(!X[S/X]).\mathsf{mclo}(X,[S/X])}\\ &= \overline{\mu X.!S.X} = \mu X.\overline{!S.X} = \mu X.?S.\overline{X} = \mu X.?S.X\end{aligned}$$

It turns out that the two steps—application of message closure and the computation of naive duality—can be combined into a single step, performing message closure during the process of computing the dual type. This is captured by the definition below, which constructs the dual of a type in a single pass over its abstract syntax tree.

Definition 21 (Duality with On-the-fly Message Closure) *For any session type S and sequence of substitutions σ closing for S, the session type* $\mathsf{dual}_{\mathsf{BH}}(S,\sigma)$ *is defined inductively by the following rules.*

$$\begin{aligned}
&\mathsf{dual}_{\mathsf{BH}}(\mathsf{end},\sigma) = \mathsf{end} && \mathsf{dual}_{\mathsf{BH}}(X,\sigma) = X\\
&\mathsf{dual}_{\mathsf{BH}}(!T.S,\sigma) = ?(T\sigma).\mathsf{dual}_{\mathsf{BH}}(S,\sigma) && \mathsf{dual}_{\mathsf{BH}}(\mu X.S,\sigma) = \mu X.\mathsf{dual}_{\mathsf{BH}}(S,[(\mu X.S)/X];\sigma)\\
&\mathsf{dual}_{\mathsf{BH}}(?T.S,\sigma) = !(T\sigma).\mathsf{dual}_{\mathsf{BH}}(S,\sigma)
\end{aligned}$$

Define $\mathsf{dual}_{\mathsf{BH}}(S)$ *as* $\mathsf{dual}_{\mathsf{BH}}(S,\varepsilon)$.

Example 22 *Here is duality with on-the-fly message closure in action for our running example $S = \mu X.!X.X$.*

$$\begin{aligned}\mathsf{dual}_{\mathsf{BH}}(S) = \mathsf{dual}_{\mathsf{BH}}(S,\varepsilon) &= \mu X.\mathsf{dual}_{\mathsf{BH}}(!X.X,[S/X])\\ &= \mu X.(?X[S/X]).\mathsf{dual}_{\mathsf{BH}}(X,[S/X])\\ &= \mu X.?S.\mathsf{dual}_{\mathsf{BH}}(X,[S/X])\\ &= \mu X.?S.X\end{aligned}$$

The economy with respect to the original definition, Example 20, should be apparent.

Example 23 *Consider the problematic type of Bernardi and Hennessy [3]. Let $S_2 = \mu Y.!Y.X$ and $S_1 = \mu X.S_2$. We then have:*

$$\begin{aligned}\mathsf{dual}_{\mathsf{BH}}(S_1) = \mathsf{dual}_{\mathsf{BH}}(S_1,\varepsilon) &= \mu X.\mathsf{dual}_{\mathsf{BH}}(S_2,[S_1/X])\\ &= \mu X.\mu Y.\mathsf{dual}_{\mathsf{BH}}(!Y.X,[S_1/X][S_2/Y])\\ &= \mu X.\mu Y.?(Y[S_1/X][S_2/Y]).\mathsf{dual}_{\mathsf{BH}}(X,[S_1/X][S_2/Y])\\ &= \mu X.\mu Y.?S_2.\mathsf{dual}_{\mathsf{BH}}(X,[S_1/X][S_2/Y])\\ &= \mu X.\mu Y.?S_2.X\end{aligned}$$

Applying message closure on the fly does not change the tail recursive type that we obtain.

Proposition 24 *If $S \in$* SType *then* $\mathsf{dual}_{\mathsf{BH}}(S)$ *is syntactically equal to* $\overline{\mathsf{mclo}(S)}$.

Proof Prove by structural induction on S that for any sequence of substitutions σ closing for S, $\mathsf{dual}_{\mathsf{BH}}(S,\sigma)$ is syntactically equal to $\mathsf{mclo}(S,\sigma)$. □

We can now show that duality with on-the-fly message closure is sound with respect to duality on regular trees.

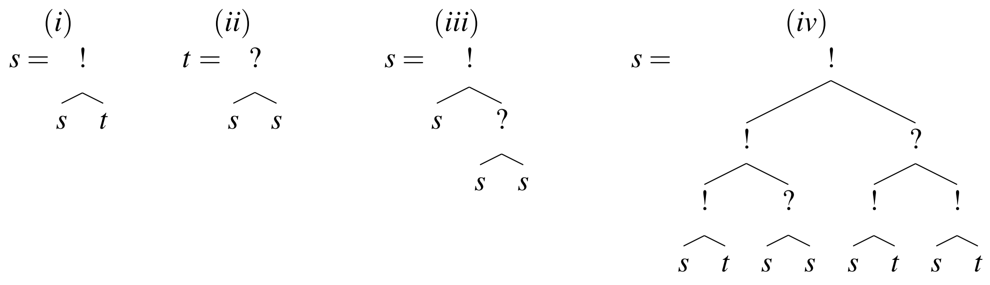
Proposition 25 *If* $S \in \mathsf{SType}$ *then* $\mathsf{treeof}(\overline{\mathsf{mclo}(S)}) \asymp \mathsf{treeof}(S)$.

Proof Instead of proving this directly, we go via definitions and results from Section 4. Proposition 24 shows that $\overline{\mathsf{mclo}(S)} = \mathsf{dual}_{\mathsf{BH}}(S)$. Proposition 36 shows that $\mathsf{dual}_{\mathsf{BH}}(S) = \mathsf{dual}_{\mathsf{LMN}}(S)$, where $\mathsf{dual}_{\mathsf{LMN}}(S)$ is defined in Definition 34. Therefore $\mathsf{treeof}(\overline{\mathsf{mclo}(S)}) = \mathsf{treeof}(\mathsf{dual}_{\mathsf{LMN}}(S))$. Finally, Proposition 37 shows that $\mathsf{treeof}(\mathsf{dual}_{\mathsf{LMN}}(S)) \asymp \mathsf{treeof}(S)$. □

4 Duality à la Lindley-Morris

The Lindley-Morris definition of the duality function [9] uses negative type variables $\overline{X}$, which we therefore add to the syntax in Definition 1. In type $\mu X.T$, both the positive variable X and the negative variable $\overline{X}$ are bound in T. Corresponding to this extension, we generalise the definition of $\mathsf{treeof}(\cdot)$ (Definition 3) so that $\mathsf{dual}(\cdot)$ (Definition 7) is applied to the subtrees that arise from negative variables.

Example 26 *Let S be the type $\mu X.!X.\overline{X}$. Let $s = \mathsf{treeof}(S)$ and let $t = \mathsf{dual}(s)$. Tree s can be depicted as (i) below. To obtain tree t, (ii) below, we dualise the root, keep s for the left subtree and use the dual of t (that is, s) for the right subtree (cf. the rule $\overline{!T.S} = ?T.\overline{S}$ in Definition 10). Substituting t into tree (i) gives tree (iii). Tree (iv) shows a few more nodes in the expansion of s.*



The definition of the duality function also requires a particular form of substitution that exchanges negative variables $\overline{X}$ and positive variables X.

Definition 27 (Negative Variable Substitution) *The result of substituting $\overline{X}$ for the free occurrences of X in T—notation $T\{\overline{X}/X\}$—is defined inductively as follows.*

$$\begin{aligned}
X\{\overline{X}/X\} &= \overline{X} & \mathsf{int}\{\overline{X}/X\} &= \mathsf{int}\\
\overline{X}\{\overline{X}/X\} &= X & \mathsf{end}\{\overline{X}/X\} &= \mathsf{end}\\
Y\{\overline{X}/X\} &= Y \quad \textit{if } Y \neq X & (?S.T)\{\overline{X}/X\} &= ?(S\{\overline{X}/X\}).T\{\overline{X}/X\}\\
\overline{Y}\{\overline{X}/X\} &= \overline{Y} & (!S.T)\{\overline{X}/X\} &= !(S\{\overline{X}/X\}).T\{\overline{X}/X\}\\
& & (\mu Y.S)\{\overline{X}/X\} &= \mu Y.S\{\overline{X}/X\}
\end{aligned}$$

Definition 28 (Lindley-Morris Duality, Original Version [9])

$$\begin{aligned}
\mathsf{dual}_{\mathsf{LM}}(\mathsf{end}) &= \mathsf{end} & \mathsf{dual}_{\mathsf{LM}}(X) &= \overline{X}\\
\mathsf{dual}_{\mathsf{LM}}(?T.S) &= !T.\mathsf{dual}_{\mathsf{LM}}(S) & \mathsf{dual}_{\mathsf{LM}}(\overline{X}) &= X\\
\mathsf{dual}_{\mathsf{LM}}(!T.S) &= ?T.\mathsf{dual}_{\mathsf{LM}}(S) & \mathsf{dual}_{\mathsf{LM}}(\mu X.S) &= \mu X.(\mathsf{dual}_{\mathsf{LM}}(S)\{\overline{X}/X\})
\end{aligned}$$

Example 29

$$\begin{aligned}\mathsf{dual}_{\mathsf{LM}}(\mu X.!X.X) &= \mu X.\mathsf{dual}_{\mathsf{LM}}((!X.X)[\overline{X}/X]) = \mu X.(?X.\mathsf{dual}_{\mathsf{LM}}(X))[\overline{X}/X] \\ &= \mu X.(?X.\overline{X})[\overline{X}/X] = \mu X.?X[\overline{X}/X].\overline{X}[\overline{X}/X] \\ &= \mu X.?\overline{X}.\overline{X}[\overline{X}/X] = \mu X.?\overline{X}.X\end{aligned}$$

This definition of duality is sound with respect to trees.

Proposition 30 *If $S \in \mathsf{SType}$ then* $\mathsf{dual}_{\mathsf{LM}}(S) \asymp S$.

Proof This is one of the results that we have mechanized in Agda (Section 5). □

There is an alternative formulation of Lindley-Morris duality that works with conventional substitution (Definition 2 with the additional clause $\overline{Y}[S/Z] = \overline{Y}$, i.e., no substitution for negative variables). The idea is that the (bound) occurrences of X in the dual of $\mu X.S$ are occurrences not of X (which stands for S) but of $\overline{X}$ (which stands for the dual of S). So we first substitute $\overline{X}$ for X in S and only then apply the duality function.

Definition 31 (Lindley-Morris Duality, Polished)

$$\begin{aligned}\mathsf{dual}_{\mathsf{LMP}}(\mathsf{end}) &= \mathsf{end} & \mathsf{dual}_{\mathsf{LMP}}(X) &= \overline{X} \\ \mathsf{dual}_{\mathsf{LMP}}(?T.S) &= !T.\mathsf{dual}_{\mathsf{LMP}}(S) & \mathsf{dual}_{\mathsf{LMP}}(\overline{X}) &= X \\ \mathsf{dual}_{\mathsf{LMP}}(!T.S) &= ?T.\mathsf{dual}_{\mathsf{LMP}}(S) & \mathsf{dual}_{\mathsf{LMP}}(\mu X.S) &= \mu X.\mathsf{dual}_{\mathsf{LMP}}(S[\overline{X}/X])\end{aligned}$$

Example 32

$$\begin{aligned}\mathsf{dual}_{\mathsf{LMP}}(\mu X.!X.X) &= \mu X.\mathsf{dual}_{\mathsf{LMP}}((!X.X)[\overline{X}/X]) \\ &= \mu X.\mathsf{dual}_{\mathsf{LMP}}(!\overline{X}.\overline{X}) = \mu X.?\overline{X}.\mathsf{dual}_{\mathsf{LMP}}(\overline{X}) = \mu X.?\overline{X}.X\end{aligned}$$

Proposition 33 *For any session type S,* $\mathsf{dual}_{\mathsf{LM}}(S)$ *is syntactically equal to* $\mathsf{dual}_{\mathsf{LMP}}(S)$.

Proof By structural induction on S, using a lemma that $\mathsf{dual}_{\mathsf{LMP}}$ commutes with substitution. □

If we are constructing the dual of a session type that contains no negative variables, we might want to avoid introducing negative variables when dualising a recursive type $\mu X.S$. We can achieve this by using Definition 31 and, at the end, replacing all occurrences of $\overline{X}$ (there are no bound occurrences of $\overline{X}$) by the original type $\mu X.S$.

Definition 34 (Lindley-Morris Duality, Yielding No New Negative Variables)

$$\begin{aligned}\mathsf{dual}_{\mathsf{LMN}}(\mathsf{end}) &= \mathsf{end} & \mathsf{dual}_{\mathsf{LMN}}(X) &= \overline{X} \\ \mathsf{dual}_{\mathsf{LMN}}((?T.S)) &= !T.\mathsf{dual}_{\mathsf{LMN}}(S) & \mathsf{dual}_{\mathsf{LMN}}(\overline{X}) &= X \\ \mathsf{dual}_{\mathsf{LMN}}((!T.S)) &= ?T.\mathsf{dual}_{\mathsf{LMN}}(S) & \mathsf{dual}_{\mathsf{LMN}}((\mu X.S)) &= \mu X.((\mathsf{dual}_{\mathsf{LMN}}(S[\overline{X}/X]))[\mu X.S/\overline{X}])\end{aligned}$$

Example 35

$$\begin{aligned}\mathsf{dual}_{\mathsf{LMN}}(S) = \mathsf{dual}_{\mathsf{LMN}}((\mu X.!X.X)) &= \mu X.\mathsf{dual}_{\mathsf{LMN}}(((!X.X)[\overline{X}/X]))[S/\overline{X}] \\ &= \mu X.\mathsf{dual}_{\mathsf{LMN}}((!\overline{X}.\overline{X}))[S/\overline{X}] \\ &= \mu X.(?\overline{X}.\mathsf{dual}_{\mathsf{LMN}}(\overline{X}))[S/\overline{X}] \\ &= \mu X.(?\overline{X}.X)[S/\overline{X}] = \mu X.?S.X\end{aligned}$$

This version of the Lindley-Morris definition coincides with the Bernardi-Hennessy definition.

Proposition 36 *For any session type S,* $\mathsf{dual}_{\mathsf{BH}}(S)$ *is syntactically equal to* $\mathsf{dual}_{\mathsf{LMN}}(S)$.

Proof If σ is a sequence of substitutions $[T_1/X_1]\dots[T_n/X_n]$ then let $\overline{\sigma} = [T_1/\overline{X_1}]\dots[T_n/\overline{X_n}]$ and $\hat{\sigma} = [\overline{X_1}/X_1]\dots[\overline{X_n}/X_n]$. Prove by structural induction on S that for any sequence of substitutions σ closing for S, $\mathsf{dual}_{\mathsf{BH}}(S,\sigma) = \mathsf{dual}_{\mathsf{LMN}}((S\hat{\sigma}))\overline{\sigma}$. The result follows by taking $\sigma = \varepsilon$. □

Finally, the Lindley-Morris definition is sound with respect to regular trees.

Proposition 37 *If* $S \in \mathsf{SType}$ *then* $\mathsf{treeof}(\mathsf{dual}_{\mathsf{LMN}}(S)) \asymp \mathsf{treeof}(S)$.

Proof First show that $\mathscr{D} = \{(S, \mathsf{dual}_{\mathsf{LMN}}(S)) \mid S \in \mathsf{SType}\}$ is a session duality (Definition 8). This establishes $\mathsf{dual}_{\mathsf{LMN}}(S) \perp S$. Then use Proposition 9. □

The substitutions in Definition 34, or equivalently in the definition of message closure (Definition 21) increase the size of the type. A simple example shows that this increase can be at least quadratic. If $S = \mu X.?X.\cdots?X.X$ with n inputs, so that the size of S is $n+2$, then $\mathsf{mclo}(S) = \mu X.?S.\cdots?S.X$ of size $n(n+2)+2$. In contrast, Definitions 28 and 31 preserve the size of the type because they only substitute variables for variables. In an implementation of a programming language with session types, it is possible to avoid computational issues resulting from these syntactic size increases, by working with a graph representation of regular trees.

5 Mechanized Results

We mechanized some of the results of the paper in Agda and are working towards a full mechanized account of all results. For accessibility, we paraphrase the definitions in standard mathematical notation rather than Agda syntax. Cognoscenti may explore the Agda source code corresponding to the development in this section in file `Duality.agda` at `https://github.com/peterthiemann/dual-session`.

The baseline for the mechanization is the coinductive formalization of session types (Definition 38), which we consider as the ground truth. In this setting, a session type is a potentially infinite tree as contained in the greatest fixpoint SType^∞ of function S_{gen}.

Definition 38 (Coinductive Session Types)

$$S_{\mathsf{gen}}(\mathscr{S}) = \{\mathsf{end}\} \cup \{!T.S, ?T.S \mid S \in \mathscr{S}, T \in \{\mathsf{int}\} \cup \mathscr{S}\}$$

Defining duality for coinductive session types is a straightforward corecursively defined function which we call $\mathsf{dual}(\cdot)$, reusing the name from Definition 7 because it implements that function.

Definition 39 (Corecursive Duality Function)

$$\mathsf{dual}(\mathsf{end}) = \mathsf{end} \qquad \mathsf{dual}(!T.S) = ?T.\mathsf{dual}(S) \qquad \mathsf{dual}(?T.S) = !T.\mathsf{dual}(S)$$

It is also straightforward to define the duality relation (cf. Def. 8) as the greatest fixpoint $(\perp)$ of $F_\perp(\cdot)$.

Definition 40 (Duality on Coinductive Session Types) *If $\mathscr{D}$ is a binary relation on tree types, then*

$$F_\perp(\mathscr{D}) = \{(\mathsf{end},\mathsf{end})\} \cup \{(!T.S, ?T.S^\perp), (?T.S, !T.S^\perp) \mid (S, S^\perp) \in \mathscr{D}\}$$

Given these definitions, it is easy to show that the corecursive duality function is sound and complete with respect to the duality relation (cf. Proposition 9).

Proposition 41 $S \perp S'$ *if and only if* $S' = \mathsf{dual}(S)$.

To formalize session types inductively, we insist that μ-types are in normal form where there are no consecutive μ-abstractions, i.e., no subterms of the form $\mu X.\mu Y.S$, and the body of a μ is never a variable. Normal forms are contractive by construction and every contractive session type (according to Definition 1) can be converted to its equivalent normal form by repeatedly coalescing subterms of the form $\mu X.\mu Y.S$ to $\mu X.S[X/Y]$ and transforming subterms of the form $\mu X.Y$ to Y, assuming $X \neq Y$. The Agda formalization enforces normal forms using two mutually recursive syntactic categories, S and S', for session types:

$$S ::= S' \mid \mu X.S' \mid X \mid \overline{X} \qquad S' ::= \mathsf{end} \mid !T.S \mid ?T.S \qquad T ::= \mathsf{int} \mid S$$

For this representation, we state various definitions of duality as shown in Sections 3 and 4. Next, we define an embedding $\lfloor\cdot\rfloor$ from SType to tree types by unfolding the recursion. This function corresponds to the $\mathsf{treeof}(\cdot)$ function (Definition 3).

$$\lfloor \mu X.S' \rfloor = \lfloor S'[\mu X.S'/X] \rfloor' \quad \lfloor \mathsf{end} \rfloor' = \mathsf{end} \quad \lfloor !T.S \rfloor' = !\lfloor T \rfloor.\lfloor S \rfloor \quad \lfloor ?T.S \rfloor' = ?\lfloor T \rfloor.\lfloor S \rfloor \quad \lfloor \mathsf{int} \rfloor = \mathsf{int}$$

This definition is mutually recursive ($\lfloor\cdot\rfloor$ applies to S and $\lfloor\cdot\rfloor'$ applies to S') and it is guarded (i.e., it yields a proper, potentially infinite term) because $\lfloor\cdot\rfloor'$ always yields a top-level constructor.

We successfully mechanised a range of results from this paper among them Proposition 30, restated here with the embedding function.

Proposition 42 *For all* $S \in \mathsf{SType}$, $\mathsf{dual}(\lfloor S \rfloor) = \lfloor \mathsf{dual}_{\mathsf{LM}}(S) \rfloor$.

6 Conclusion

We surveyed the competing definitions of session type duality in the presence of recursion. Starting from an interpretation of session types as trees, and a duality relation on trees, we establish soundness of the Bernardi-Hennessy and the Lindley-Morris definitions of duality on syntactic session types. We further come up with streamlined versions of these definitions and justify the original flawed definition of duality (naive duality) when restricted to tail recursive session types. We have mechanized some results in Agda, and are working on mechanizing the others.

In summary, we have tied up the remaining loose ends in the definition of duality of session types. Many of the issues in prior work are caused by syntax, namely by reliance on μ-types to express recursion. Taking a standard interpretation of recursive types as regular trees, and the corresponding formalization by coinductive definitions in Agda, is effective in proving the soundness of syntactic definitions.
Acknowledgements. Simon Gay was partially supported by the UK EPSRC grant EP/K034413/1 "From Data Types to Session Types: A Basis for Concurrency and Distribution" and by the EU Horizon 2020 MSCA-RISE project 778233 "BehAPI: Behavioural Application Program Interfaces". Vasco T. Vasconcelos was supported by FCT through the LASIGE Research Unit, ref. UIDB/00408/2020, and by COST Action CA15123 EUTypes. We thank Sam Lindley and Garrett Morris for discussions.

References

[1] Hendrik Pieter Barendregt (1985): *The Lambda Calculus — its Syntax and Semantics*. *Studies in Logic and the Foundations of Mathematics* 103, North-Holland.

[2] Giovanni Bernardi, Ornela Dardha, Simon J. Gay & Dimitrios Kouzapas (2014): *On Duality Relations for Session Types*. In: *TGC*, doi:10.1007/978-3-662-45917-1_4.

[3] Giovanni Bernardi & Matthew Hennessy (2014): *Using Higher-Order Contracts to Model Session Types (Extended Abstract)*. In: *CONCUR*, doi:10.1007/978-3-662-44584-6_27.

[4] Giovanni Bernardi & Matthew Hennessy (2016): *Using higher-order contracts to model session types*. *Logical Methods in Computer Science* 12(2), doi:10.2168/LMCS-12(2:10)2016.

[5] Simon J. Gay & Malcolm Hole (1999): *Types and Subtypes for Client-Server Interactions*. In: *ESOP*, doi:10.1007/3-540-49099-X_6.

[6] Simon J. Gay & Malcolm Hole (2005): *Subtyping for session types in the pi calculus*. *Acta Informatica* 42(2-3), doi:10.1007/s00236-005-0177-z.

[7] Kohei Honda (1993): *Types for Dyadic Interaction*. In: *CONCUR*, doi:10.1007/3-540-57208-2_35.

[8] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In: *ESOP*, doi:10.1007/BFb0053567.

[9] Sam Lindley & J. Garrett Morris (2016): *Talking bananas: structural recursion for session types*. In: *ICFP*, doi:10.1145/2951913.2951921.

[10] Benjamin C. Pierce (2002): *Types and Programming Languages*. MIT Press.

[11] Kaku Takeuchi, Kohei Honda & Makoto Kubo (1994): *An Interaction-based Language and its Typing System*. In: *PARLE*, doi:10.1007/3-540-58184-7_118.

[12] Vasco T. Vasconcelos (2012): *Fundamentals of session types*. *Information and Computation* 217, pp. 52–70, doi:10.1016/j.ic.2012.05.002.