A Logical Account of Subtyping for Session Types

Ross Horne
University of Luxembourg

Luca Padovani
University of Camerino

We study the notion of subtyping for session types in a logical setting, where session types are propositions of multiplicative/additive linear logic extended with least and greatest fixed points. The resulting subtyping relation admits a simple characterization that can be roughly spelled out as the following lapalissade: every session type is larger than the smallest session type and smaller than the largest session type. At the same time, we observe that this subtyping, unlike traditional ones, preserves termination in addition to the usual safety properties of sessions. We present a calculus of sessions that adopts this subtyping relation and we show that subtyping, while useful in practice, is superfluous in the theory: every use of subtyping can be "compiled away" via a coercion semantics.

1 Introduction

Session types [12, 13, 15] are descriptions of communication protocols supported by an elegant correspondence with linear logic [23, 3, 16] that provides session type systems with solid logical foundations. As an example, below is the definition of a session type describing the protocol implemented by a mathematical server (in the examples of this section, $\&$ and $\oplus$ are n-ary operators denoting external and internal labeled choices, respectively):

$$B = \&\{\mathsf{end} : \bot, \mathsf{add} : \mathsf{Num}^\bot \parr \mathsf{Num}^\bot \parr \mathsf{Num} \otimes B\}$$

According to the session type B, the server first waits for a label – either end or add – that identifies the operation requested by the client. If the label is end, the client has no more requests and the server terminates. If the label is add, the server waits for two numbers, sends their sum back to the client and then makes itself available again offering the same protocol B. In this example, we write $\mathsf{Num}^\bot$ for the type of numbers being consumed and Num for the type of numbers being produced. A client of this server could implement a communication protocol described by the following session type:

$$A = \oplus\{\mathsf{add} : \mathsf{Num} \otimes \mathsf{Num} \otimes \mathsf{Num}^\bot \parr \oplus\{\mathsf{end} : \mathbf{1}\}\}$$

This client sends the label add followed by two numbers, it receives the result and then terminates the interaction with the server by sending the label end. When we connect two processes through a session, we expect their interaction to be flawless. In many session type systems, this is guaranteed by making sure that the session type describing the behavior of one process is the *dual* of the session type describing the behavior of its peer. *Duality*, often denoted by $\cdot^\bot$, is the operator on session types that inverts the *direction* of messages without otherwise altering the structure of protocol. In the above example it is clear that A is *not* the dual of B nor is B the dual of A. Nonetheless, we would like such client and such server to be declared compatible, since the client is exercising only a subset of the capabilities of the server. To express this compatibility we have to resort to a more complex relation between A and B, either by observing that B (the behavior of the server) is a *more accommodating* version of $A^\bot$ or by observing that A (the behavior of the client) is a *less demanding* version of $B^\bot$. We make these relations precise by means of a *subtyping relation* $\leqslant$ for session types. Subtyping enhances the applicability of

type systems by means of the well-known substitution principle: an entity of type C can be used where an entity of type D is expected if C is a subtype of D. After the initial work of Gay and Hole [9] many subtyping relations for session types have been studied [4, 20, 17, 21, 10]. Such subtyping relations differ widely in the way they are defined and/or in the properties they preserve, but they all share the fact that subtyping is essentially defined by the branching structure of session types given by labels. To illustrate this aspect, let us consider again the session types A and B defined above. We have

$$B \leqslant \&\{\mathsf{add} : \mathsf{Num}^\perp \parr \mathsf{Num}^\perp \parr \mathsf{Num} \otimes \&\{\mathsf{end} : \perp\}\} = A^\perp \tag{1}$$

meaning that a server behaving as B can be safely used where a server behaving as $A^\perp$ is expected. Dually, we also have

$$A \leqslant \oplus\{\mathsf{end} : \mathbf{1}, \mathsf{add} : \mathsf{Num} \otimes \mathsf{Num} \otimes \mathsf{Num}^\perp \parr B^\perp\} = B^\perp \tag{2}$$

meaning that a client behaving as A can be safely used where a client behaving as $B^\perp$ is expected. Note how subtyping is crucially determined by the sets of labels that can be received/sent when comparing two related types. In (1), the server of type B is willing to accept any label from the set $\{\mathsf{end}, \mathsf{add}\}$, which is a *superset* of $\{\mathsf{add}\}$ that we have in $A^\perp$. In (2), the client is (initially) sending a label from the set $\{\mathsf{add}\}$, which is a subset of $\{\mathsf{end}, \mathsf{add}\}$ that we have in $B^\perp$. This co/contra variance of labels in session types is a key distinguishing feature of all known notions of subtyping for session types.[1]

In this work we study the notion of subtyping for session types in a setting where session types are propositions of $\mu\mathsf{MALL}^\infty$ [2, 6], the infinitary proof theory of multiplicative additive linear logic extended with least and greatest fixed points. Our investigation has two objectives. First, to understand whether and how it is possible to capture the well-known co/contra variance of behaviors when the connectives used to describe branching session types ($\&$ and $\oplus$ of linear logic) have fixed arity. Second, to understand whether there are criticial aspects of subtyping that become relevant when typing derivations are meant to be logically sound.

At the core of our proposal is the observation that, when session types (hence process behaviors) are represented by linear logic propositions [23, 3, 16], it is impossible to write a process that behaves as $\mathbf{0}$ and it is very easy to write a process that behaves as $\top$. If we think of a session type as the set of processes that behave according to that type, this means that the additive constants $\mathbf{0}$ and $\top$ may serve well as the least and greatest elements of a session subtyping relation. Somewhat surprisingly, the subtyping relation defined by these properties of $\mathbf{0}$ and $\top$ allows us to express essentially the same subtyping relations that arise from the usual co/contra variance of labels. For example, following our proposal the session type of the client, previously denoted A, would instead be written as

$$C = \oplus\{\mathsf{end} : \mathbf{0}, \mathsf{add} : \mathsf{Num} \otimes \mathsf{Num} \otimes \mathsf{Num}^\perp \parr \oplus\{\mathsf{end} : \mathbf{1}, \mathsf{add} : \mathbf{0}\}\}$$

using which we can derive both

$$B \leqslant \&\{\mathsf{end} : \top, \mathsf{add} : \mathsf{Num}^\perp \parr \mathsf{Num}^\perp \parr \mathsf{Num} \otimes \&\{\mathsf{end} : \perp, \mathsf{add} : \top\}\} = C^\perp \qquad \text{as well as} \qquad C \leqslant B^\perp$$

without comparing labels and just using the fact that $\mathbf{0}$ is the least session type and $\top$ the greatest one. Basically, instead of *omitting those labels* that correspond to impossible continuations (*cf.* the missing

[1] Gay and Hole [9] and other authors [4, 20, 21] define subtyping for session types in such a way that the *opposite* relations of eqs. (1) and (2) hold. Both viewpoints are viable depending on whether session types are considered to be types of *channels* or types of *processes*. Here we take the latter stance, referring to Gay [8] for a comparison of the two approaches.

Process	$P,Q ::=$			$\mid (x)(P \mid Q)$	composition	
	$\mathsf{A}\langle \overline{x} \rangle$	invocation		$\mid \mathsf{fail}\ x$	failure	
	$\mid\ x().P$	signal input		$\mid x[]$	signal output	
	$\mid\ x(y).P$	channel input		$\mid x[y](P \mid Q)$	channel output	
	$\mid\ \mathsf{case}\ x\{P,Q\}$	choice input		$\mid x[\mathsf{in}_i].P$	choice output	$i \in \{0,1\}$

Table 1: Syntax of $\mu\mathsf{CP}^{\infty}$.

end and add in A), we use the uninhabited session type $\mathbf{0}$ or its dual $\top$ *as impossible continuations* (*cf.* C). It could be argued that the difference between the two approaches is mostly cosmetic. Indeed, it is easy to devise (de)sugaring functions to rewrite session types from one syntax to the other. However, the novel approach we propose allows us to recast the well-known subtyping relation for session types in a logical setting. A first consequence of this achievement is that the soundness of the type system *with subtyping* does not require an *ad hoc* proof, but follows from the soundness of the type system *without subtyping* through a suitable coercion semantics. In addition, we find out that the subtyping relation we propose preserves not only the usual *safety properties* – communication safety, protocol fidelity and deadlock freedom – but also *termination*, which is a *liveness property*.

Structure of the paper. In Section 2 we define $\mu\mathsf{CP}^{\infty}$, a session calculus of processes closely related to $\mu\mathsf{CP}$ [16] and CP [23]. In Section 3 we define the type language for $\mu\mathsf{CP}^{\infty}$ and the subtyping relation. In Section 4 we define the typing rules for $\mu\mathsf{CP}^{\infty}$ and give a coercion semantics to subtyping, thus showing that the type system of $\mu\mathsf{CP}^{\infty}$ is a conservative extension of $\mu\mathsf{MALL}^{\infty}$ [2, 6]. We wrap up in Section 5.

2 Syntax and semantics of $\mu\mathsf{CP}^{\infty}$

The syntax of $\mu\mathsf{CP}^{\infty}$ is shown in Table 1 and makes use of a set of *process names* A, B, ... and of an infinite set of *channels* x, y, z and so on. The calculus includes standard forms representing communication actions: $\mathsf{fail}\ x$ models a process failing on x; $x().P$ and $x[]$ model the input/output of a termination signal on x; $\mathsf{case}\ x\{P,Q\}$ and $x[\mathsf{in}_i].P$ model the input/output of a label in_i on x; $x(y).P$ and $x[y](P \mid Q)$ model the input/output of a channel y on x. Note that $x[y](P \mid Q)$ outputs a *new* channel y which is bound in P but not in Q. Free channel output can be encoded as shown in previous works [16]. The form $(x)(P \mid Q)$ models a session x connecting two parallel processes P and Q and the form $\mathsf{A}\langle \overline{x} \rangle$ models the invocation of the process named A with parameters $\overline{x}$. For each process name A we assume that there is a unique global definition of the form $\mathsf{A}(\overline{x}) \triangleq P$ that gives its meaning. Hereafter $\overline{x}$ denotes a possibly empty sequence of channels. The notions of free and bound channels are defined in the expected way. We identify processes up to renaming of bound channels and we write $\mathsf{fn}(P)$ for the set of free channels of P.

The operational semantics of $\mu\mathsf{CP}^{\infty}$ is shown in Table 2 and consists of a structural pre-congruence relation $\preccurlyeq$ and a reduction relation $\rightarrow$, both of which are fairly standard. We write $P \rightarrow$ if $P \rightarrow Q$ for some Q and we say that P is *stuck*, notation $P \not\rightarrow$, if not $P \rightarrow$.

Example 2.1. We can model client and server described in Section 1 as the processes below.

$$\mathsf{Client}(x) \triangleq x[\mathsf{in}_1].x[\mathsf{in}_0].x[] \qquad \mathsf{Server}(x,z) \triangleq \mathsf{case}\ x\{x().z[], \mathsf{Server}\langle x,z \rangle\}$$

For simplicity, we only focus on the overall structure of the processes rather than on the actual mathematical operations they perform, so we omit any exchange of concrete data from this model. ⌟

$$
\begin{array}{rrcll}
\text{[S-PAR-COMM]} & (x)(P \mid Q) & \preccurlyeq & (x)(Q \mid P) & \\
\text{[S-PAR-ASSOC]} & (x)(P \mid (y)(Q \mid R)) & \preccurlyeq & (y)((x)(P \mid Q) \mid R) & x \in \mathsf{fn}(Q) \setminus \mathsf{fn}(R), y \notin \mathsf{fn}(P) \\
\text{[S-CALL]} & \mathsf{A}\langle \overline{x} \rangle & \preccurlyeq & P & \mathsf{A}(\overline{x}) \triangleq P \\
 & & & & \\
\text{[R-CLOSE]} & (x)(x[] \mid x().P) & \to & P & \\
\text{[R-COMM]} & (x)(x[y](P \mid Q) \mid x(y).R) & \to & (y)(P \mid (x)(Q \mid R)) & \\
\text{[R-CASE]} & (x)(x[\mathsf{in}_i].P \mid \mathsf{case}\ x\{Q_0, Q_1\}) & \to & (x)(P \mid Q_i) & i \in \{0,1\} \\
\text{[R-PAR]} & (x)(P \mid R) & \to & (x)(Q \mid R) & P \to Q \\
\text{[R-STRUCT]} & P & \to & Q & P \preccurlyeq P' \to Q' \preccurlyeq Q
\end{array}
$$

Table 2: Structrual pre-congruence and reduction semantics of $\mu\mathsf{CP}^{\infty}$.

We conclude this section with the definitions of the properties ensured by our type system, namely *deadlock freedom* and *termination*. The latter notion is particularly relevant in our setting since termination preservation is a novel aspect of the subtyping relation that we are about to define.

Definition 2.1 (deadlock-free process)**.** We say that P is *deadlock free* if $P \Rightarrow Q \not\to$ implies that Q is not (structurally pre-congruent to) a process of the form $(x)(R_1 \mid R_2)$.

A deadlock-free process either reduces or it is stuck waiting to synchronize on some free channel.

Definition 2.2 (terminating process)**.** A *run* of a process P is a (finite or infinite) sequence $(P_0, P_1, \dots)$ of processes such that $P_0 = P$ and $P_i \to P_{i+1}$ whenever $i+1$ is a valid index of the sequence. We say that a run is maximal if either it is infinite or if the last process in it is stuck. We say that P is *terminating* if every maximal run of P is finite.

Note that a terminating process is not necessarily free of restrictions. For example, $(x)(\mathsf{fail}\ x \mid x[])$ is terminated but not deadlock free. It really is the conjunction of deadlock freedom and termination (as defined above) that ensure that a process is "well behaved".

3 Types and subtyping

The type language for $\mu\mathsf{CP}^{\infty}$ consists of the propositions of $\mu\mathsf{MALL}^{\infty}$ [2, 6, 1], the infinitary proof theory of multiplicative/additive linear logic extended with least and greatest fixed points. We start from the definition of *pre-types*, which are linear logic propositions built using type variables taken from an infinite set and ranged over by X and Y.

$$\textbf{Pre-type} \qquad A, B ::= X \mid \bot \mid \mathbf{1} \mid \top \mid \mathbf{0} \mid A \parr B \mid A \otimes B \mid A \,\&\, B \mid A \oplus B \mid \nu X.A \mid \mu X.A$$

The usual notions of free and bound type variables apply. A *type* is a closed pre-type. We assume that type variables occurring in types are *guarded*. That is, we forbid types of the form $\sigma_1 X_1 \dots \sigma_n X_n . X_i$ where $\sigma_1, \dots, \sigma_n \in \{\mu, \nu\}$. We write $A^{\perp}$ for the *dual* of A, which is defined in the expected way with the proviso that $X^{\perp} = X$. This way of dualizing type variables is not problematic since we will always apply $\cdot^{\perp}$ to types, which contain no free type variables. As usual, we write $A\{B/X\}$ for the (pre-)type obtained by replacing every X occurring free in the pre-type A with the type B. Hereafter we let κ range over the constants $\mathbf{0}$, $\mathbf{1}$, $\bot$ and $\top$, we let $\star$ range over the connectives $\&$, $\oplus$, $\parr$ and $\otimes$ and σ range over the binders μ and ν. Also, we say that any type of the form $\sigma X.A$ is a σ-type.

$$\frac{}{\mathbf{0} \leqslant A}\ [\text{BOT}] \qquad \frac{}{A \leqslant \top}\ [\text{TOP}] \qquad \frac{}{\kappa \leqslant \kappa}\ [\text{REFL}] \qquad \frac{A \leqslant A' \qquad B \leqslant B'}{A \star B \leqslant A' \star B'}\ [\text{CONG}]$$

$$\frac{A\{\sigma X.A/X\} \leqslant B}{\sigma X.A \leqslant B}\ [\text{LEFT-}\sigma] \qquad \frac{A \leqslant B\{\sigma X.B/X\}}{A \leqslant \sigma X.B}\ [\text{RIGHT-}\sigma]$$

Table 3: Subtyping for session types.

We write $\preceq$ for the standard *sub-formula* relation on types. To be precise, the relation $\preceq$ is the least preorder on types such that $A \preceq \sigma X.A$ and $A_i \preceq A_1 \star A_2$. For example, consider $A \stackrel{\text{def}}{=} \mu X.\nu Y.(1 \oplus X)$ and its unfolding $A' \stackrel{\text{def}}{=} \nu Y.(1 \oplus A)$. We have $A \preceq 1 \oplus A \preceq A'$, hence A is a sub-formula of A'. Given a set $\mathscr{T}$ of types we write $\min \mathscr{T}$ for the $\preceq$-minimum type in $\mathscr{T}$ when it is defined.

Table 3 shows the inference rules for subtyping judgments. The rules are meant to be interpreted coinductively so that a judgment $A \leqslant B$ is derivable if it is the conclusion of a finite/infinite derivation. The rules [BOT] and [TOP] establish that $\mathbf{0}$ and $\top$ are respectively the least and the greatest session type; the rules [REFL] and [CONG] establish reflexivity and pre-congruence of $\leqslant$ with respect to all the constants and connectives; the rules [LEFT-σ] and [RIGHT-σ] allow fixed points to be unfolded on either side of $\leqslant$.

Example 3.1. Consider the types $A \stackrel{\text{def}}{=} \mathbf{0} \oplus (\mathbf{1} \oplus \mathbf{0})$ and $B \stackrel{\text{def}}{=} \nu X.(\bot \,\&\, X)$ which, as we will see later, describe the behavior of Client and Server in Example 2.1. We can derive both $A \leqslant B^\perp$ and $B \leqslant A^\perp$ thus:

$$\dfrac{\dfrac{}{\mathbf{0} \leqslant \mathbf{1}}\ [\text{BOT}] \qquad \dfrac{\dfrac{\dfrac{}{\mathbf{1} \leqslant \mathbf{1}}\ [\text{REFL}] \qquad \dfrac{}{\mathbf{0} \leqslant B^\perp}\ [\text{BOT}]}{\mathbf{1} \oplus \mathbf{0} \leqslant \mathbf{1} \oplus B^\perp}\ [\text{CONG}]}{\mathbf{1} \oplus \mathbf{0} \leqslant B^\perp}\ [\text{RIGHT-}\mu]}{\dfrac{A \leqslant \mathbf{1} \oplus B^\perp}{A \leqslant B^\perp}\ [\text{RIGHT-}\mu]}\ [\text{CONG}]$$

$$\dfrac{\dfrac{}{\bot \leqslant \top}\ [\text{TOP}] \qquad \dfrac{\dfrac{\dfrac{}{\bot \leqslant \bot}\ [\text{REFL}] \qquad \dfrac{}{B \leqslant \top}\ [\text{TOP}]}{\bot \parr B \leqslant \bot \parr \top}\ [\text{CONG}]}{B \leqslant \bot \,\&\, \top}\ [\text{LEFT-}\nu]}{\dfrac{\bot \parr B \leqslant A^\perp}{B \leqslant A^\perp}\ [\text{LEFT-}\nu]}\ [\text{CONG}]$$

The rules [LEFT-σ] and [RIGHT-σ] may look suspicious since they are applicable to either side of $\leqslant$ regardless of the intuitive interpretation of μ and ν as least and greatest fixed points. In fact, if subtyping were solely defined by the derivability according to the rules in Table 3, the two fixed point operators would be equivalent. For example, both $\mu X.(\mathbf{1} \oplus X) \leqslant \nu X.(\mathbf{1} \oplus X)$ and $\nu X.(\mathbf{1} \oplus X) \leqslant \mu X.(\mathbf{1} \oplus X)$ are derivable even though only the first relation seems reasonable. We will see in Example 4.2 that allowing the second relation is actually *unsound*, in the sense that it compromises the termination property enjoyed by well-typed processes. We obtain a sound subtyping relation by ruling out some infinite derivations as per the following (and final) definition of subtyping.

Definition 3.1 (subtyping)**.** We say that A is a *subtype* of B if $A \leqslant B$ is derivable and, for every infinite branch $(A_i \leqslant B_i)_{i \in \mathbb{N}}$ of the derivation, either (1) $\min\{C \mid \exists^\infty i : A_i = C\}$ is a μ-type or (2) $\min\{C \mid \exists^\infty i : B_i = C\}$ is a ν-type. Hereafter $\exists^\infty i$ means the existence of infinitely many i's with the stated property.

The clauses (1) and (2) of Definition 3.1 make sure that μ and ν are correctly interpreted as least and greatest fixed points. In particular, we expect the least fixed point to be subsumed by a greatest fixed point, but not vice versa in general. For example, consider once again the (straightforward) derivations for the aforementioned subtyping judgments $\mu X.(\mathbf{1} \oplus X) \leqslant \nu X.(\mathbf{1} \oplus X)$ and $\nu X.(\mathbf{1} \oplus X) \leqslant \mu X.(\mathbf{1} \oplus X)$. The first derivation satisfies both clauses (there is only one infinite branch, along which a μ-type is unfolded infinitely many times on the left hand side of $\leqslant$ and a ν-type is unfolded infinitely many times on the right hand side of $\leqslant$). The second derivation satisfies neither clause. Therefore, $\mu X.(\mathbf{1} \oplus X)$ is a

$$\text{[CALL]}\ \frac{P \vdash \overline{x : A}}{\mathsf{A}\langle \overline{x} \rangle \vdash \overline{x : A}}\ \mathsf{A}(\overline{x}) \triangleq P \qquad \text{[SUB]}\ \frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, x : B}{(x)(P \mid Q) \vdash \Gamma, \Delta}\ A \leqslant B^{\perp} \qquad \text{[}\top\text{]}\ \frac{}{\mathsf{fail}\ x \vdash \Gamma, x : \top} \qquad \text{[}\perp\text{]}\ \frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \perp}$$

$$\text{[}\mathbf{1}\text{]}\ \frac{}{x[] \vdash x : \mathbf{1}} \qquad \text{[}\parr\text{]}\ \frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B} \qquad \text{[}\otimes\text{]}\ \frac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{x[y](P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B} \qquad \text{[\&]}\ \frac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B}{\mathsf{case}\ x\{P, Q\} \vdash \Gamma, x : A \,\&\, B}$$

$$\text{[}\oplus\text{]}\ \frac{P \vdash \Gamma, x : A_i}{x[\mathsf{in}_i].P \vdash \Gamma, x : A_0 \oplus A_1}\ i \in \{0, 1\} \qquad \text{[}\sigma\text{]}\ \frac{P \vdash \Gamma, x : A\{\sigma X.A/X\}}{P \vdash \Gamma, x : \sigma X.A}$$

Table 4: Typing rules for $\mu\mathsf{CP}^{\infty}$.

subtype of $\nu X.(\mathbf{1} \oplus X)$ but $\nu X.(\mathbf{1} \oplus X)$ is not a subtype of $\mu X.(\mathbf{1} \oplus X)$. As we will see in Section 4, the application of a subtyping relation $A \leqslant B$ can be explicitly modeled as a process *consuming* a channel of type A while *producing* a channel of type B. According to this interpretation of subtyping, we can see that clause (1) of Definition 3.1 is just a dualized version of clause (2).

In both clauses of Definition 3.1 there is a requirement that the type of the fixed point on each side of the relation is determined by the $\preceq$-minimum of the types that appear infinitely often on either side. This is needed to handle correctly alternating fixed points, by determining which one is actively contributing to the infinite path. To see what effect this has consider the types $A \stackrel{\text{def}}{=} \mu X.\nu Y.(\mathbf{1} \oplus X)$, $A' \stackrel{\text{def}}{=} \nu Y.(\mathbf{1} \oplus A)$, $B \stackrel{\text{def}}{=} \mu X.\mu Y.(\mathbf{1} \oplus X)$ and $B' \stackrel{\text{def}}{=} \mu Y.(\mathbf{1} \oplus B)$. Observe that A unfolds to A', A' unfolds to $\mathbf{1} \oplus A$, B unfolds to B' and B' unfolds to $\mathbf{1} \oplus B$. We have $A \leqslant B$ despite Y is bound by a *greatest* fixed point on the left and by a *least* fixed point on the right. Indeed, both A and A' occur infinitely often in the (only) infinite branch of the derivation for $A \leqslant B$, but $A \preceq A'$ according to the intuition that the $\preceq$-minimum type that occurs infinitely often is the one corresponding to the outermost fixed point. In this case, the outermost fixed point is μX which "overrides" the contribution of the inner fixed point νY. The interested reader may refer to the literature on $\mu\mathsf{MALL}^{\infty}$ [2, 6] for details.

Hereafter, unless otherwise specified, we write $A \leqslant B$ to imply that A is a subtype of B and not simply that the judgment $A \leqslant B$ is derivable. It is possible to show that $\leqslant$ is a preorder and that $A \leqslant B$ implies $B^{\perp} \leqslant A^{\perp}$. Indeed, as illustrated in Example 3.1, we obtain a derivation of $B^{\perp} \leqslant A^{\perp}$ from that of $A \leqslant B$ by dualizing every judgment and by turning every application of [LEFT-σ] (respectively [RIGHT-σ], [BOT], [TOP]) into an application of [RIGHT-$\sigma^{\perp}$] (respectively [LEFT-$\sigma^{\perp}$], [TOP], [BOT]).

4 Typing rules

In this section we describe the typing rules for $\mu\mathsf{CP}^{\infty}$. Typing judgments have the form $P \vdash \Gamma$ where P is a process and Γ is a typing context, namely a finite map from channels to types. We can read this judgment as the fact that P behaves as described by the types in the range of Γ with respect to the channels in the domain of Γ. We write $\mathsf{dom}(\Gamma)$ for the domain of Γ, we write $x : A$ for the typing context with domain $\{x\}$ that maps x to A, we write Γ, Δ for the union of Γ and Δ when $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset$. The typing rules of $\mu\mathsf{CP}^{\infty}$ are shown in Table 4 and, with the exception of [CALL] and [SUB], they correspond to the proof

rules of $\mu\mathsf{MALL}^\infty$ [2, 6] in which the context is the sequent being proved and the process is (almost) a syntactic representation of the proof. The rules for the multiplicative/additive constants and for the connectives are standard. The rule $[\sigma]$ where $\sigma \in \{\mu, \nu\}$ simply unfolds fixed points regardless of their nature. The rule [CALL] unfolds a process invocation into its definition, checking that the invocation and the definition are well typed in the same context. Finally, [SUB] checks that the composition $(x)(P \mid Q)$ is well typed provided that A (the behavior of P with respect to x) is a subtype of $B^\perp$ (where B is the behavior of Q with respect to x). In this sense [SUB] embeds the substitution principle induced by $\leqslant$ since it allows a process behaving as A to be used where a process behaving as $B^\perp$ is expected. Note that the standard cut rule of $\mu\mathsf{MALL}^\infty$ is a special case of [SUB] because of the reflexivity of $\leqslant$.

Like in $\mu\mathsf{MALL}^\infty$, the rules are meant to be interpreted coinductively so that a judgment $P \vdash \Gamma$ is deemed derivable if there is an arbitrary (finite or infinite) derivation whose conclusion is $P \vdash \Gamma$.

Example 4.1. Let us show the typing derivations for the processes discussed in Example 2.1. To this aim, let $A \stackrel{\text{def}}{=} \mathbf{0} \oplus (\mathbf{1} \oplus \mathbf{0})$ and $B \stackrel{\text{def}}{=} \nu X.(\perp \,\&\, X)$ and recall from Example 3.1 that $A \leqslant B^\perp$. We derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x[] \vdash \mathbf{1}}\,[\mathbf{1}]}{x[\mathsf{in}_0].x[] \vdash \mathbf{1} \oplus \mathbf{0}}\,[\oplus]}{x[\mathsf{in}_1].x[\mathsf{in}_0].x[] \vdash x : A}\,[\oplus]}{\mathsf{Client}\langle x\rangle \vdash x : A}\,[\text{CALL}] \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{z[] \vdash z : \mathbf{1}}\,[\mathbf{1}]}{x().z[] \vdash x : \perp, z : \mathbf{1}}\,[\perp] \quad \dfrac{\vdots}{\mathsf{Server}\langle x,z\rangle \vdash x : B, z : \mathbf{1}}}{\mathsf{case}\ x\{x().z[], \mathsf{Server}\langle x,z\rangle\} \vdash x : \perp \,\&\, B, z : \mathbf{1}}\,[\&]}{\mathsf{case}\ x\{x().z[], \mathsf{Server}\langle x,z\rangle\} \vdash x : B, z : \mathbf{1}}\,[\nu]}{\mathsf{Server}\langle x,z\rangle \vdash x : B, z : \mathbf{1}}\,[\text{CALL}]$$

$$\dfrac{\mathsf{Client}\langle x\rangle \vdash x : A \qquad \mathsf{Server}\langle x,z\rangle \vdash x : B, z : \mathbf{1}}{(x)(\mathsf{Client}\langle x\rangle \mid \mathsf{Server}\langle x,z\rangle) \vdash z : \mathbf{1}}\,[\text{SUB}]$$

We can obtain a similar typing derivation by swapping Client and Server and using the relation $B \leqslant A^\perp$. Note that Client and Server cannot be composed directly using a standard cut since $A \neq B^\perp$. So, the use of subtyping in the above typing derivation is important to obtain a well-typed composition. ⌟

It is a known fact that not every $\mu\mathsf{MALL}^\infty$ derivation is a valid one [2, 6, 1]. In order to characterize the valid derivations we need some auxiliary notions which we recall below.

Definition 4.1 (thread)**.** Let $\gamma = (P_i \vdash \Gamma_i)_{i\in\mathbb{N}}$ be an infinite branch in a typing derivation and recall that $P_{i+1} \vdash \Gamma_{i+1}$ is a premise of $P_i \vdash \Gamma_i$. A *thread* of γ is a sequence $(x_i)_{i\geq k}$ of channels such that $x_i \in \mathsf{dom}(\Gamma_i)$ and either $x_i = x_{i+1}$ or $P_i = x_i[x_{i+1}](P_{i+1} \mid Q)$ or $P_i = x_i(x_{i+1}).P_{i+1}$ for every $i \geq k$.

Intuitively, a thread is an infinite sequence of channel names $(x_i)_{i\geq k}$ that are found starting from some position k in an infinite branch $(P_i \vdash \Gamma_i)_{i\in\mathbb{N}}$ and that pertain to the same session. For example, consider the derivation in Example 4.1 and observe that there is only one infinite branch, the rightmost one. The sequence $(x,x,x,\dots)$ is a thread that starts right above the conclusion of the derivation.

Definition 4.2 (ν-thread)**.** Given a branch $\gamma = (P_i \vdash \Gamma_i)_{i\in\mathbb{N}}$ and a thread $t = (x_i)_{i\geq k}$ of γ, we write $\mathsf{inf}(\gamma,t) \stackrel{\text{def}}{=} \{A \mid \exists^\infty i \geq k : \Gamma_i(x_i) = A\}$. We say that t is a *ν-thread* of γ if $\min \mathsf{inf}(\gamma,t)$ is a ν-type.

Given a branch $\gamma = (P_i \vdash \Gamma_i)_{i\in\mathbb{N}}$ and a thread $t = (x_i)_{i\geq k}$ of γ, the thread identifies an infinite sequence $(\Gamma_i(x_i))_{i\geq k}$ of types. The set $\mathsf{inf}(\gamma,t)$ is the set of those types that occur infinitely often in this sequence and $\min \mathsf{inf}(\gamma,t)$ is the $\preceq$-minimum among these types (it can be shown that the minimum of any set $\mathsf{inf}(\gamma,t)$ is always defined [6]). We say that t is a ν-thread if such minimum type is a ν-type. In Example 4.1, the thread $t = (x,x,x,\dots)$ identifies the sequence $(B,B,\perp\,\&\,B,B,\dots)$ of types in which both B and $\perp\,\&\,B$ occur infinitely often. Since $B \preceq \perp\,\&\,B$ and B is a ν-type we conclude that t is a ν-thread.

$$\left[\!\!\left[\frac{}{\mathbf{0} \leqslant A}\right]\!\!\right]_{x,y} \triangleq \mathsf{fail}\ x \qquad \left[\!\!\left[\frac{}{\mathbf{1} \leqslant \mathbf{1}}\right]\!\!\right]_{x,y} \triangleq x().y[]$$

$$\left[\!\!\left[\frac{\pi_1 :: A \leqslant A' \qquad \pi_2 :: B \leqslant B'}{A \oplus B \leqslant A' \oplus B'}\right]\!\!\right]_{x,y} \triangleq \mathsf{case}\ x\{y[\mathsf{in}_0].[\![\pi_1]\!]_{x,y}, y[\mathsf{in}_1].[\![\pi_2]\!]_{x,y}\}$$

$$\left[\!\!\left[\frac{\pi_1 :: A \leqslant A' \qquad \pi_2 :: B \leqslant B'}{A \otimes B \leqslant A' \otimes B'}\right]\!\!\right]_{x,y} \triangleq x(u).y[v]([\![\pi_1]\!]_{u,v} \mid [\![\pi_2]\!]_{x,y}) \quad (u \text{ and } v \text{ fresh})$$

$$\left[\!\!\left[\frac{\pi :: A\{\sigma X.A/X\} \leqslant B}{\sigma X.A \leqslant B}\right]\!\!\right]_{x,y} \triangleq [\![\pi]\!]_{x,y} \qquad \left[\!\!\left[\frac{\pi :: A \leqslant B\{\sigma X.B/X\}}{A \leqslant \sigma X.B}\right]\!\!\right]_{x,y} \triangleq [\![\pi]\!]_{x,y}$$

Table 5: Coercion semantics of subtyping (selected equations).

Definition 4.3 (valid branch)**.** Let $\gamma = (P_i \vdash \Gamma_i)_{i\in\mathbb{N}}$ be an infinite branch of a typing derivation. We say that γ is *valid* if there is a ν-thread $(x_i)_{i\geq k}$ of γ such that $[\nu]$ is applied to infinitely many of the x_i.

Definition 4.3 establishes that a branch is valid if it contains a ν-thread in which the ν-type occurring infinitely often is also unfolded infinitely often. This happens in Example 4.1, in which the $[\nu]$ rule is applied infinitely often to unfold the type of x. The reader familiar with the μMALL$^\infty$ literature may have spotted a subtle difference between our notion of valid branch and the standard one [2, 6]. In μMALL$^\infty$, a branch is valid only provided that the ν-thread in it is not "eventually constant", namely if the greatest fixed point that defines the ν-thread is unfolded infinitely many times. This condition is satisfied by our notion of valid branch because of the requirement that there must be infinitely many applications of $[\nu]$ concerning the names in the ν-thread. Now we can define the notion of valid typing derivation.

Definition 4.4 (valid derivation)**.** A typing derivation is *valid* if so is every infinite branch in it.

Following Pierce [22] we provide a *coercion semantics* to our subtyping relation by means of two translation functions, one on derivations of subtyping relations $A \leqslant B$ and one on typing derivations $P \vdash \Gamma$ that make use of subtyping. The first translation is (partially) given in Table 5. The translation takes a derivation π of a subtyping relation $A \leqslant B$ – which we denote by $\pi :: A \leqslant B$ – and generates a process $[\![\pi]\!]_{x,y}$ that transforms (the protocol described by) A into (the protocol described by) B. The translation is parametrized by the two channels x and y on which the transformation takes place: the protocol A is "consumed" from x and reissued on y as a protocol B. In Table 5 we show a fairly complete selection of cases, the remaining ones being obvious variations. It is easy to establish that $[\![\pi]\!]_{x,y} \vdash x : A^\perp, y : B$ if $A \leqslant B$. In particular, consider an infinite branch $\gamma \stackrel{\text{def}}{=} ([\![\pi_i]\!]_{x_i,y_i} \vdash x_i : A_i^\perp, y : B_i)_{i\in\mathbb{N}}$ in the typing derivation of the coercion where $A_0 = A$ and $B_0 = B$. This branch corresponds to an infinite branch $(A_i \leqslant B_i)_{i\in\mathbb{N}}$ in $\pi :: A \leqslant B$. According to Definition 3.1, either clause (1) or clause (2) holds for this branch. Suppose, without loss of generality, that clause (1) holds. Then $\min\{C \mid \exists^\infty i \in \mathbb{N} : A_i = C\}$ is a μ-type. According to Table 5 we have that $(x_i)_{i\in\mathbb{N}}$ is a ν-thread of γ, hence γ is a valid branch. Note that in general $[\![\pi]\!]_{x,y}$ is (the invocation of) a recursive process.

Concerning the translation of typing derivations, it is defined by the equation

$$\left[\!\!\left[\dfrac{\pi_1 :: P \vdash \Gamma, x : A \qquad \pi_2 :: Q \vdash \Gamma, x : B}{(x)(P \mid Q) \vdash \Gamma} \right]\!\!\right] = \dfrac{\dfrac{[\![\pi_1\{y/x\}]\!] \qquad [\![\pi]\!]_{y,x} \vdash y : A^{\perp}, x : B}{(y)(P\{y/x\} \mid [\![\pi]\!]_{y,x}) \vdash \Gamma, x : B} \qquad [\![\pi_2]\!]}{(x)((y)(P\{y/x\} \mid [\![\pi]\!]_{y,x}) \mid Q) \vdash \Gamma} \tag{3}$$

where $\pi :: A \leqslant B^{\perp}$ and extended homomorphically to all the other typing rules in Table 4. Note that (3) turns every application of the [SUB] into two applications of the standard μMALL$^{\infty}$ cut rule. The validity of the resulting typing derivation follows immediately from that of the original typing derivation and that for the coercion, as argued earlier.

Thanks to the correspondence between μCP$^{\infty}$'s typing rules and μMALL$^{\infty}$, well-typed μCP$^{\infty}$ processes are well behaved. In particular, processes that are well typed in a singleton context are deadlock free.

Theorem 4.1 (deadlock freedom)**.** *If $P \vdash x : A$ then P is deadlock free.*

Moreover, the cut elimination property of μMALL$^{\infty}$ [2, 6] can be used to prove that well-typed μCP$^{\infty}$ processes terminate, similarly to related systems [16, 5].

Theorem 4.2 (termination)**.** *If $P \vdash \Gamma$ then P is terminating.*

Proof sketch. The typing derivation for $P \vdash \Gamma$ with the subtype coercion made explicit maps directly to a valid μMALL$^{\infty}$ proof. Every reduction step of P maps directly to one or more principal reductions in the μMALL$^{\infty}$ proof. The reason why we could have more than one principal reduction for each process reduction comes from our choice of not having an explicit process form triggering the unfolding of a fixed point (see $[\sigma]$). Now, suppose that P has an infinite run. Then there would be an infinite sequence of reduction steps starting from P, hence an infinite sequence of cut reductions in the corresponding μMALL$^{\infty}$ proof, which contradicts [6, Proposition 3.5]. Thus every run of P must be finite. □

Note that Theorem 4.2 only assures that a well-typed process will not reduce forever, not necessarily that the final configuration of the process is free of restricted sessions. These may occur guarded by a prefix concerning some free channel in the process. We can formulate a property of "successful termination" by combining Theorems 4.1 and 4.2.

Corollary 4.1. *If $P \vdash x : \mathbf{1}$ then P eventually reduces to $x[]$.*

We conclude this section with an example showing that the additional clauses of Definition 3.1 are key to making sure that $\leqslant$ is a termination-preserving subtyping relation.

Example 4.2. Consider a degenerate client $\mathsf{Chatter}(x) \triangleq x[\mathsf{in}_1].\mathsf{Chatter}\langle x\rangle$ that engages into an infinite interaction with Server from Example 2.1 and let $C \stackrel{\text{def}}{=} \nu X.(\mathbf{1} \oplus X)$. The derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{Chatter}\langle x\rangle \vdash x : C}\,[\nu]}{x[\mathsf{in}_1].\mathsf{Chatter}\langle x\rangle \vdash x : \mathbf{1} \oplus C}\,[\oplus]}{\mathsf{Chatter}\langle x\rangle \vdash x : \mathbf{1} \oplus C}\,[\text{CALL}]}{\mathsf{Chatter}\langle x\rangle \vdash x : C}\,[\nu]$$

is valid since the only infinite branch contains a ν-thread $(x, x, \dots)$ along which we find infinitely many applications of $[\nu]$. If we allowed the relation $C \leqslant B^{\perp}$ (*cf.* the discussion leading to Definition 3.1) the composition $(x)(\mathsf{Chatter}\langle x\rangle \mid \mathsf{Server}\langle x, z\rangle)$ would be well typed and it would no longer be the case that well-typed processes terminate, as the interaction between Chatter and Server goes on forever. ⌟

5 Concluding remarks

We have defined a subtyping relation for session types as the precongruence that is insensitive to the (un)folding of recursive types and such that $\mathbf{0}$ and $\top$ act as least and greatest elements. Despite the minimalistic look of the relation and the apparent rigidity in the syntax of types, in which the arity of internal and external choices is fixed, $\leqslant$ captures the usual co/contra variance of labels thanks to the interpretation given to $\mathbf{0}$ and $\top$. Other refinement relations for session types with least and greatest elements have been studied in the past [19, 21], although without an explicit correspondance with logic.

Unlike subtyping relations for session types [9, 4, 17, 10] that only preserve *safety properties* of sessions (communication safety, protocol fidelity and deadlock freedom), $\leqslant$ also preserves termination, which is a *liveness property*. For this reason, $\leqslant$ is somewhat related to *fair subtyping* [20, 21], which preserves *fair termination* [11, 7]. It appears that $\leqslant$ is coarser than fair subtyping, although the exact relationship between the two relations is difficult to characterize because of the fundamentally different ways in which recursive behaviors are represented in the syntax of types. The subtyping relation defined in this paper inherits least and greatest fixed points from $\mu\mathsf{MALL}^{\infty}$ [2, 6], whereas fair subtyping has been studied on session type languages that either make use of general recursion [20] or that use regular trees directly [21]. A more conclusive comparison is left for future work.

A key difference between the treatment of fixed points in this work and a related logical approach to session subtyping [14] is that, while both guarantee deadlock freedom, the current approach also guarantees termination. Insight concerning the design of fixed points should be exportable to other session calculi independently from any logical interpretation. In particular, it would be interesting to study subtyping for *asynchronous session types* [17, 10] in light of Definition 3.1. This can be done by adopting a suitable coercion semantics to enable buffering of messages as in simple orchestrators [18].

Acknowledgments. We are grateful to the anonymous reviewers for their thoughtful comments.

References

[1] David Baelde, Amina Doumane, Denis Kuperberg & Alexis Saurin (2022): *Bouncing Threads for Circular and Non-Wellfounded Proofs: Towards Compositionality with Circular Proofs*. In Christel Baier & Dana Fisman, editors: *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, ACM, pp. 63:1–63:13, doi:10.1145/3531130.3533375.

[2] David Baelde, Amina Doumane & Alexis Saurin (2016): *Infinitary Proof Theory: the Multiplicative Additive Case*. In Jean-Marc Talbot & Laurent Regnier, editors: *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, *LIPIcs* 62, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 42:1–42:17, doi:10.4230/LIPIcs.CSL.2016.42.

[3] Luís Caires, Frank Pfenning & Bernardo Toninho (2016): *Linear logic propositions as session types*. *Math. Struct. Comput. Sci.* 26(3), pp. 367–423, doi:10.1017/S0960129514000218.

[4] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino & Luca Padovani (2009): *Foundations of session types*. In António Porto & Francisco Javier López-Fraguas, editors: *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, ACM, pp. 219–230, doi:10.1145/1599410.1599437.

[5] Farzaneh Derakhshan & Frank Pfenning (2022): *Circular Proofs as Session-Typed Processes: A Local Validity Condition*. *Logical Methods in Computer Science* Volume 18, Issue 2, doi:10.46298/lmcs-18(2:8)2022.

[6] Amina Doumane (2017): *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. Ph.D. thesis, Paris Diderot University, France. Available at `https://tel.archives-ouvertes.fr/tel-01676953`.

[7] Nissim Francez (1986): *Fairness*. Monographs in Comp. Sci., Springer, doi:10.1007/978-1-4612-4886-6.

[8] Simon J. Gay (2016): *Subtyping Supports Safe Session Substitution*. In Sam Lindley, Conor McBride, Philip W. Trinder & Donald Sannella, editors: *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, *Lecture Notes in Computer Science* 9600, Springer, pp. 95–108, doi:10.1007/978-3-319-30936-1_5.

[9] Simon J. Gay & Malcolm Hole (2005): *Subtyping for session types in the pi calculus*. *Acta Informatica* 42(2-3), pp. 191–225, doi:10.1007/s00236-005-0177-z.

[10] Silvia Ghilezan, Jovanka Pantović, Ivan Prokić, Alceste Scalas & Nobuko Yoshida (2022): *Precise Subtyping for Asynchronous Multiparty Sessions*. *ACM Trans. Comput. Logic*, doi:10.1145/3568422. Just Accepted.

[11] Orna Grumberg, Nissim Francez & Shmuel Katz (1984): *Fair Termination of Communicating Processes*. In: *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, Association for Computing Machinery, New York, NY, USA, pp. 254–265, doi:10.1145/800222.806752.

[12] Kohei Honda (1993): *Types for Dyadic Interaction*. In Eike Best, editor: *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, *Lecture Notes in Computer Science* 715, Springer, pp. 509–523, doi:10.1007/3-540-57208-2_35.

[13] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In Chris Hankin, editor: *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Lisbon, Portugal, March 28 - April 4*, *Lecture Notes in Computer Science* 1381, Springer, pp. 122–138, doi:10.1007/BFb0053567.

[14] Ross Horne (2020): *Session Subtyping and Multiparty Compatibility Using Circular Sequents*. In Igor Konnov & Laura Kovács, editors: *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, *LIPIcs* 171, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 12:1–12:22, doi:10.4230/LIPIcs.CONCUR.2020.12.

[15] Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira & Gianluigi Zavattaro (2016): *Foundations of Session Types and Behavioural Contracts*. *ACM Comput. Surv.* 49(1), pp. 3:1–3:36, doi:10.1145/2873052.

[16] Sam Lindley & J. Garrett Morris (2016): *Talking bananas: structural recursion for session types*. In Jacques Garrigue, Gabriele Keller & Eijiro Sumii, editors: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, ACM, pp. 434–447, doi:10.1145/2951913.2951921.

[17] Dimitris Mostrous & Nobuko Yoshida (2015): *Session typing and asynchronous subtyping for the higher-order π-calculus*. *Inf. Comput.* 241, pp. 227–263, doi:10.1016/j.ic.2015.02.002.

[18] Luca Padovani (2010): *Contract-based discovery of Web services modulo simple orchestrators*. *Theor. Comput. Sci.* 411(37), pp. 3328–3347, doi:10.1016/j.tcs.2010.05.002.

[19] Luca Padovani (2010): *Session Types = Intersection Types + Union Types*. In Elaine Pimentel, Betti Venneri & Joe B. Wells, editors: *Proceedings Fifth Workshop on Intersection Types and Related Systems, ITRS 2010, Edinburgh, U.K., 9th July 2010*, *EPTCS* 45, pp. 71–89, doi:10.4204/EPTCS.45.6.

[20] Luca Padovani (2013): *Fair Subtyping for Open Session Types*. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska & David Peleg, editors: *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, *Lecture Notes in Computer Science* 7966, Springer, pp. 373–384, doi:10.1007/978-3-642-39212-2_34.

[21] Luca Padovani (2016): *Fair subtyping for multi-party session types*. *Math. Struct. Comput. Sci.* 26(3), pp. 424–464, doi:10.1017/S096012951400022X.

[22] Benjamin C. Pierce (2002): *Types and programming languages*. MIT Press.

[23] Philip Wadler (2014): *Propositions as sessions*. *J. Funct. Program.* 24(2-3), pp. 384–418, doi:10.1017/S095679681400001X.