Precise subtyping for synchronous multiparty sessions *

Mariangiola Dezani-Ciancaglini
Università di Torino, Italy†

Silvia Ghilezan
Univerzitet u Novom Sadu, Serbia

Svetlana Jakšić
Univerzitet u Novom Sadu, Serbia

Jovanka Pantović
Univerzitet u Novom Sadu, Serbia

Nobuko Yoshida
Imperial College London‡

The notion of subtyping has gained an important role both in theoretical and applicative domains: in lambda and concurrent calculi as well as in programming languages. The soundness and the completeness, together referred to as the preciseness of subtyping, can be considered from two different points of view: operational and denotational. The former preciseness has been recently developed with respect to type safety, i.e. the safe replacement of a term of a smaller type when a term of a bigger type is expected. The latter preciseness is based on the denotation of a type which is a mathematical object that describes the meaning of the type in accordance with the denotations of other expressions from the language. The result of this paper is the operational and denotational preciseness of the subtyping for a synchronous multiparty session calculus. The novelty of this paper is the introduction of characteristic global types to prove the operational completeness.

1 Introduction

In modelling distributed systems, where many processes interact by means of message passing, one soon realises that most interactions are meant to occur within the scope of private channels according to disciplined protocols. Following [13], we call such private interactions *multiparty sessions* and the protocols that describe them *multiparty session types*.

The ability to describe complex interaction protocols by means of a formal, simple and yet expressive type language can have a profound impact on the way distributed systems are designed and developed. This is witnessed by the fact that some important standardisation bodies for web-based business and finance protocols [2, 22, 20] have recently investigated design and implementation frameworks for specifying message exchange rules and validating business logic based on the notion of multiparty sessions, where multiparty session types are "shared agreements" between teams of programmers developing possibly large and complex distributed protocols or software systems.

Subtyping has been extensively studied as one of the most interesting issues in type theory. The correctness of subtyping relations has been usually provided as the operational soundness: If T is a subtype of T' (notation $\mathsf{T} \leq \mathsf{T}'$), then a term of type T may be provided whenever a term of type T' is needed, see [19] (Chapter 15) and [9] (Chapter 23). The converse direction, the operational completeness, has been largely ignored in spite of its usefulness to define the greatest subtyping relation ensuring type safety. If $[\![\mathsf{T}]\!]$ is the set interpretating type T, then a subtyping is denotationally sound when $\mathsf{T} \leq \mathsf{T}'$ implies $[\![\mathsf{T}]\!] \subseteq [\![\mathsf{T}']\!]$ and denotationally complete when $[\![\mathsf{T}]\!] \subseteq [\![\mathsf{T}']\!]$ implies $\mathsf{T} \leq \mathsf{T}'$. *Preciseness* means both soundness and completeness.

*Partly supported by COST IC1201 BETTY and DART bilateral project between Italy and Serbia.
†Partly supported by MIUR PRIN Project CINA Prot. 2010LHT4KM and Torino University/Compagnia San Paolo Project SALT.
‡Partly supported by EPSRC EP/K011715/1, EP/K034413/1, and EP/L00058X/1, and EU Project FP7-612985 UpScale.

S. J. Gay and J. Alglave (Eds.): Programming Language Approaches to Concurrency- and Communication-Centric Software (PLACES 2015).
EPTCS 203, 2016, pp. 29–43, doi:10.4204/EPTCS.203.3

Operational preciseness has been first introduced in [16] for a call-by-value λ-calculus with sum, product and recursive types. Both operational and denotational preciseness have been studied in [7] for a λ-calculus with choice and parallel constructors [6] and in [3] for binary sessions [21].

These facts ask for investigating precise subtyping for multiparty session types, the subject of this paper. Subtyping for session calculi can be defined to assure safety of substitutability of either channels [8] or processes [5]. We claim that substitutability of processes better fits the notion of preciseness.

We show the operational and denotational preciseness of the subtyping introduced in [5] for a simplification of the synchronous multiparty session calculus in [15]. For the operational preciseness we take the view that well-typed sessions never get stuck. For the denotational preciseness we interpret a type as the set of processes having that type.

The most technical challenge is the operational completeness, which requires a non trivial extension of the method used in the case of binary sessions. The core of this extension is the construction of *characteristic global types*.

Outline The calculus and its type system are introduced in Sections 2 and 3, respectively. Sections 4 and 6 contain the proofs of operational and denotational preciseness. Section 5 illustrates the operational preciseness by means of an example. Some concluding remarks are the content of Section 7.

2 Synchronous Multiparty Session Calculus

This section introduces syntax and semantics of a synchronous multiparty session calculus. Since our focus is on subtyping, we simplify the calculus in [15] eliminating both shared channels for session initiations and session channels for communications inside sessions. We conjecture the preciseness of the subtyping in [5] also for the full calculus, but we could not use the present approach for the proof, since well-typed interleaved sessions can be stuck [4].

Syntax A *multiparty session* is a series of interactions between a fixed number of participants, possibly with branching and recursion, and serves as a unit of abstraction for describing communication protocols.

We use the following base sets: *values*, ranged over by $\mathsf{v},\mathsf{v}',\ldots$; *expressions*, ranged over by $\mathsf{e},\mathsf{e}',\ldots$; *expression variables*, ranged over by $x,y,z\ldots$; *labels*, ranged over by $\ell,\ell',\ldots$; *session participants*, ranged over by $\mathsf{p},\mathsf{q},\ldots$; *process variables*, ranged over by $X,Y,\ldots$; *processes*, ranged over by $P,Q,\ldots$; and *multiparty sessions*, ranged over by $\mathscr{M},\mathscr{M}',\ldots$.

The values are natural numbers n, integers i, and boolean values true and false. The expressions e are variables or values or expressions built from expressions by applying the operators $\mathtt{succ},\mathtt{neg},\neg,\oplus$, or the relation $>$. An *evaluation context* $\mathscr{E}$ is an expression with exactly one hole, built in the same manner from expressions and the hole.

Processes P are defined by:

$$P \quad ::= \quad \mathsf{p}?\ell(x).P \;\mid\; \mathsf{p}!\ell(\mathsf{e}).P \;\mid\; P+P \;\mid\; \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ P \;\mid\; \mu X.P \;\mid\; X \;\mid\; \mathbf{0}$$

The input process $\mathsf{p}?\ell(x).P$ waits for an expression with label ℓ from participant p and the output process $\mathsf{q}!\ell(\mathsf{e}).Q$ sends the value of expression e with label ℓ to participant q. The external choice $P+Q$ offers to choose either P or Q. The process $\mu X.P$ is a recursive process. We take an equi-recursive view, not distinguishing between a process $\mu X.P$ and its unfolding $P\{\mu X.P/X\}$. We assume that the recursive processes are guarded, i.e. $\mu X.X$ is not a process.

A multiparty session $\mathscr{M}$ is a parallel composition of pairs (denoted by $\mathsf{p}\triangleleft P$) of participants and processes:

$$\mathscr{M} \quad ::= \quad \mathsf{p}\triangleleft P \;\mid\; \mathscr{M} \mid \mathscr{M}$$

$$\mathtt{succ}(\mathsf{n}) \downarrow (\mathsf{n}+1) \quad \mathtt{neg}(\mathsf{i}) \downarrow (-\mathsf{i}) \quad \neg\mathsf{true} \downarrow \mathsf{false} \quad \neg\mathsf{false} \downarrow \mathsf{true} \quad \mathsf{v} \downarrow \mathsf{v}$$

$$(\mathsf{i}_1 > \mathsf{i}_2) \downarrow \begin{cases} \mathsf{true} & \text{if } \mathsf{i}_1 > \mathsf{i}_2, \\ \mathsf{false} & \text{otherwise} \end{cases} \qquad \frac{\mathsf{e}_1 \downarrow \mathsf{v} \text{ or } \mathsf{e}_2 \downarrow \mathsf{v}}{\mathsf{e}_1 \oplus \mathsf{e}_2 \downarrow \mathsf{v}} \qquad \frac{\mathsf{e} \downarrow \mathsf{v} \quad \mathscr{E}(\mathsf{v}) \downarrow \mathsf{v}'}{\mathscr{E}(\mathsf{e}) \downarrow \mathsf{v}'}$$

Table 1: Expression evaluation.

$$\text{[S-EXTCH 1]}\ P+Q \equiv Q+P \qquad \text{[S-EXTCH 2]}\ (P+Q)+R \equiv P+(Q+R) \qquad \text{[S-MULTI]}\ P \equiv Q \Rightarrow \mathsf{p} \triangleleft P \equiv \mathsf{p} \triangleleft Q$$

$$\text{[S-PAR 1]}\ \mathsf{p} \triangleleft \mathbf{0} \mid \mathscr{M} \equiv \mathscr{M} \qquad \text{[S-PAR 2]}\ \mathscr{M} \mid \mathscr{M}' \equiv \mathscr{M}' \mid \mathscr{M} \qquad \text{[S-PAR 3]}\ (\mathscr{M} \mid \mathscr{M}') \mid \mathscr{M}'' \equiv \mathscr{M} \mid (\mathscr{M}' \mid \mathscr{M}'')$$

Table 2: Structural congruence.

We will use $\sum_{i \in I} P_i$ as short for $P_1 + \ldots + P_n$, and $\prod_{i \in I} \mathsf{p}_i \triangleleft P_i$ as short for $\mathsf{p}_1 \triangleleft P_1 \mid \ldots \mid \mathsf{p}_n \triangleleft P_n$, where $I = \{1, \ldots, n\}$.

If $\mathsf{p} \triangleleft P$ is well typed (see Table 8), then participant p does not occur in process P, since we do not allow self-communications.

Operational semantics *The value* v *of expression* e (notation $\mathsf{e} \downarrow \mathsf{v}$) is as expected, see Table 1. The successor operation `succ` is defined only on natural numbers, the negation `neg` is defined on integers (and then also on natural numbers), and $\neg$ is defined only on boolean values. The internal choice $\mathsf{e}_1 \oplus \mathsf{e}_2$ evaluates either to the value of e_1 or to the value of e_2.

The *computational rules of multiparty sessions* (Table 3) are closed with respect to the structural congruence defined in Table 2 and the following reduction contexts:

$$\mathscr{C}[\cdot] ::= [\cdot] \parallel \mathscr{C}[\cdot] \mid \mathscr{M}$$

In rule [R-COMM] participant q sends the value v choosing label ℓ_j to participant p which offers inputs on all labels ℓ_i with $i \in I$. We use $\longrightarrow^*$ with the standard meaning.

In order to define the operational preciseness of subtyping it is crucial to formalise when a multiparty session contains communications that will never be executed.

Definition 2.1 *A multiparty session* $\mathscr{M}$ *is* stuck *if* $\mathscr{M} \not\equiv \mathsf{p} \triangleleft \mathbf{0}$ *and there is no multiparty session* $\mathscr{M}'$ *such that* $\mathscr{M} \longrightarrow \mathscr{M}'$. *A multiparty session* $\mathscr{M}$ *gets* stuck, *notation* $\mathtt{stuck}(\mathscr{M})$, *if it reduces to a stuck multiparty session.*

$$\text{[R-COMM]}\ \frac{j \in I \qquad \mathsf{e} \downarrow \mathsf{v}}{\mathsf{p} \triangleleft \sum_{i \in I} \mathsf{q}?\ell_i(x).P_i \mid \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).Q \longrightarrow \mathsf{p} \triangleleft P_j\{\mathsf{v}/x\} \mid \mathsf{q} \triangleleft Q} \qquad \text{[T-CONDITIONAL]}\ \frac{\mathsf{e} \downarrow \mathsf{true}}{\mathsf{p} \triangleleft \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow \mathsf{p} \triangleleft P}$$

$$\text{[F-CONDITIONAL]}\ \frac{\mathsf{e} \downarrow \mathsf{false}}{\mathsf{p} \triangleleft \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow \mathsf{p} \triangleleft Q} \qquad \text{[R-CONTEXT]}\ \frac{\mathscr{M} \longrightarrow \mathscr{M}'}{\mathscr{C}[\mathscr{M}] \longrightarrow \mathscr{C}[\mathscr{M}']} \qquad \text{[R-STRUCT]}\ \frac{\mathscr{M}_1' \equiv \mathscr{M}_1 \quad \mathscr{M}_1 \longrightarrow \mathscr{M}_2 \quad \mathscr{M}_2 \equiv \mathscr{M}_2'}{\mathscr{M}_1' \longrightarrow \mathscr{M}_2'}$$

Table 3: Reduction rules.

$$\mathsf{T} \between \mathsf{T}' = \begin{cases} \mathsf{T} & \text{if } \mathsf{T} = \mathsf{T}', \\ \mathsf{T} \wedge \mathsf{T}' & \text{if } \mathsf{T} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \text{ and } \mathsf{T}' = \bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathsf{T}'_j \\ & \text{ and } \ell_i \neq \ell'_j \text{ for all } i \in I, j \in J \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I} \upharpoonright \mathsf{r} = \begin{cases} \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{r} & \text{if } \mathsf{r} = \mathsf{p}, \\ \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{r} & \text{if } \mathsf{r} = \mathsf{q}, \\ \between_{i\in I} \mathsf{G}_i \upharpoonright \mathsf{r} & \text{if } \mathsf{r} \neq \mathsf{p}, \mathsf{r} \neq \mathsf{q} \text{ and } \between_{i\in I} \mathsf{G}_i \upharpoonright \mathsf{r} \text{ is defined.} \end{cases}$$

$$(\mu \mathbf{t}.\mathsf{G}) \upharpoonright \mathsf{r} = \begin{cases} \mu \mathbf{t}.\mathsf{G} \upharpoonright \mathsf{r} & \text{if } \mathsf{r} \text{ occurs in } \mathsf{G}, \\ \mathtt{end} & \text{otherwise.} \end{cases} \qquad \mathbf{t} \upharpoonright \mathsf{r} = \mathbf{t} \qquad \mathtt{end} \upharpoonright \mathsf{r} = \mathtt{end}$$

Table 4: Projection of global types onto participants.

3 Type System

This section introduces the type system, which is a simplification of that in [15] due to the new formulation of the calculus.

Types *Sorts* are ranged over by S and defined by: $S ::= \mathtt{nat} \mid \mathtt{int} \mid \mathtt{bool}$
Global types generated by:

$$\mathsf{G} ::= \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I} \mid \mu \mathbf{t}.\mathsf{G} \mid \mathbf{t} \mid \mathtt{end}$$

describe the whole conversation scenarios of multiparty sessions. *Session types* correspond to projections of global types on the individual participants. Inspired by [18], we use intersection and union types instead of standard branching and selection [13] to take advantage from the subtyping induced by subset inclusion. The grammar of session types, ranged over by T, is then

$$\mathsf{T} ::= \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \mid \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i \mid \mu \mathbf{t}.\mathsf{T} \mid \mathbf{t} \mid \mathtt{end}$$

We require that $\ell_i \neq \ell_j$ with $i \neq j$ and $i, j \in I$ and recursion to be guarded in both global and session types. Recursive types with the same regular tree are considered equal [19, Chapter 20, Section 2]. In writing types we omit unnecessary brackets, intersections, unions and `end`.

We extend the original definition of projection of global types onto participants [13] in the line of [23], but keeping the definition simpler than that of [23]. This generalisation is enough to project the characteristic global types of next Section. We use the partial operator $\between$ on session types. This operator applied to two identical types gives one of them, applied to two intersection types with same sender and different labels gives their intersection and it is undefined otherwise, see Table 4. The same table gives the *projection* of the global type G onto the participant r, notation $\mathsf{G} \upharpoonright \mathsf{r}$. This projection allows participants to receive different messages in different branches of global types.

Example 3.1 *If* $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{nat}).\mathsf{G}_1, \ell_2(\mathtt{bool}).\mathsf{G}_2\}$, *where*
$\mathsf{G}_1 = \mathsf{q} \to \mathsf{r} : \ell_3(\mathtt{int})$ *and* $\mathsf{G}_2 = \mathsf{q} \to \mathsf{r} : \ell_5(\mathtt{nat})$ *and* $\mathsf{r} \neq \mathsf{p}$, *then*

$$\mathsf{G} \upharpoonright \mathsf{r} = \mathsf{G}_1 \upharpoonright \mathsf{r} \between \mathsf{G}_2 \upharpoonright \mathsf{r} = \mathsf{q}?\ell_3(\mathtt{int}) \between \mathsf{q}?\ell_5(\mathtt{nat}) = \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat}).$$

Subtyping *Subsorting* $\leq:$ on sorts is the minimal reflexive and transitive closure of the relation induced by the rule: $\mathtt{nat} \leq: \mathtt{int}$. *Subtyping* $\leqslant$ on session types takes into account the contra-variance of inputs, the covariance of outputs, and the standard rules for intersection and union. Table 5 gives the subtyping rules: the double line in rules indicates that the rules are interpreted *coinductively* [19] (Chapter 21). Subtyping can be easily decided, see for example [8]. For reader convenience Table 6 gives the procedure

$$[\text{SUB-END}]\quad \mathtt{end} \leqslant \mathtt{end} \qquad [\text{SUB-IN}]\quad \frac{\forall i \in I: \quad S_i' \leq: S_i \quad \mathsf{T}_i \leqslant \mathsf{T}_i'}{\bigwedge_{i\in I\cup J} \mathtt{p}?\ell_i(S_i).\mathsf{T}_i \leqslant \bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i').\mathsf{T}_i'} \qquad [\text{SUB-OUT}]\quad \frac{\forall i \in I: \quad S_i \leq: S_i' \quad \mathsf{T}_i \leqslant \mathsf{T}_i'}{\bigvee_{i\in I} \mathtt{p}!\ell_i(S_i).\mathsf{T}_i \leqslant \bigvee_{i\in I\cup J} \mathtt{p}!\ell_i(S_i').\mathsf{T}_i'}$$

Table 5: Subtyping rules.

$$\mathscr{S}(\Theta,\mathsf{T},\mathsf{T}') = \begin{cases} \mathsf{true} & \text{if } \mathsf{T} \leqslant \mathsf{T}' \in \Theta \text{ or } \mathsf{T} = \mathsf{T}' \\ \&_{i\in I}\mathscr{S}(\Theta \cup \{\mathsf{T} \leqslant \mathsf{T}'\}, \mathsf{T}_i, \mathsf{T}_i') & \text{if } (\mathsf{T} = \bigwedge_{i\in I\cup J} \mathtt{p}?\ell_i(S_i).\mathsf{T}_i \text{ and } \mathsf{T}' = \bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i').\mathsf{T}_i' \\ & \quad \text{and } \forall i \in I: S_i' \leq: S_i) \text{ or} \\ & \quad (\mathsf{T} = \bigvee_{i\in I} \mathtt{p}!\ell_i(S_i).\mathsf{T}_i \text{ and } \mathsf{T}' = \bigvee_{i\in I\cup J} \mathtt{p}!\ell_i(S_i').\mathsf{T}_i' \\ & \quad \text{and } \forall i \in I: S_i \leq: S_i') \\ \mathsf{false} & \text{otherwise} \end{cases}$$

Table 6: The procedure $\mathscr{S}(\Theta,\mathsf{T},\mathsf{T}')$.

$\mathscr{S}(\Theta,\mathsf{T},\mathsf{T}')$, where Θ is a set of subtyping judgments. This procedure terminates since unfolding of session types generates regular trees, so Θ cannot grow indefinitely and we have only a finite number of subtyping judgments to consider. Clearly $\mathscr{S}(\emptyset,\mathsf{T},\mathsf{T}')$ is equivalent to $\mathsf{T} \leqslant \mathsf{T}'$.

Typing system We distinguish three kinds of typing judgments

$$\Gamma \vdash \mathtt{e} : S \qquad \Gamma \vdash P : \mathsf{T} \qquad \vdash \mathscr{M} : \mathsf{G},$$

where Γ is the environment $\Gamma ::= \emptyset \mid \Gamma, x : S \mid \Gamma, X : \mathsf{T}$ that associates expression variables with sorts and process variables with session types. The typing rules for expressions are standard, see Table 7. Table 8 gives the typing rules for processes and multiparty sessions. Processes are typed as expected, the syntax of session types only allows input processes in external choices and output processes in the branches of conditionals. We need to assure that processes in external choices offer different labels. For this reason rule [T-IN-CHOICE] types both inputs and external choices. With two separate rules:

$$\frac{\Gamma, x : S \vdash P : \mathsf{T}}{\Gamma \vdash \mathtt{q}?\ell(x).P : \mathtt{q}?\ell(S).\mathsf{T}}\ [\text{T-IN}] \qquad \frac{\Gamma \vdash P_1 : \mathsf{T}_1 \quad \Gamma \vdash P_2 : \mathsf{T}_2}{\Gamma \vdash P_1 + P_2 : \mathsf{T}_1 \wedge \mathsf{T}_2}\ [\text{T-CHOICE}]$$

we could derive

$$\vdash \mathtt{q}?\ell_1(x).\mathbf{0} + \mathtt{q}?\ell_2(x).\mathbf{0} + \mathtt{q}?\ell_1(x).\mathtt{q}!\ell_5(\mathtt{true}).\mathbf{0} : \mathtt{q}?\ell_2(\mathtt{int}).\mathtt{end} \wedge \mathtt{q}?\ell_1(\mathtt{int}).\mathtt{q}!\ell_5(\mathtt{bool}).\mathtt{end}.$$

In order to type a session, rule [T-SESS] requires that the processes in parallel can play as participants of a whole communication protocol or the terminated process, i.e. their types are projections of a unique global type. We define the set $\mathtt{pt}\{\mathsf{G}\}$ of participants of a global type G as follows:

$$\mathtt{pt}\{\mathtt{p} \to \mathtt{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\} = \{\mathtt{p},\mathtt{q}\} \cup \mathtt{pt}\{\mathsf{G}_i\}\ (i \in I)^1$$
$$\mathtt{pt}\{\mu\mathbf{t}.\mathsf{G}\} = \mathtt{pt}\{\mathsf{G}\} \qquad \mathtt{pt}\{\mathbf{t}\} = \emptyset \qquad \mathtt{pt}\{\mathtt{end}\} = \emptyset$$

The condition $\mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathtt{p}_i \mid i \in I\}$ allows to type also sessions containing $\mathtt{p} \triangleleft \mathbf{0}$, a property needed to assure invariance of types under structural congruence.

The proposed type system for multiparty sessions enjoys type preservation under reduction (subject reduction) and the safety property that a typed multiparty session will never get stuck. The remaining of this section is devoted to the proof of these properties.

[1] The projectability of G assures $\mathtt{pt}\{\mathsf{G}_i\} = \mathtt{pt}\{\mathsf{G}_j\}$ for all $i, j \in I$.

$$\Gamma \vdash \mathsf{n} : \mathtt{nat} \qquad \Gamma \vdash \mathsf{i} : \mathtt{int} \qquad \Gamma \vdash \mathsf{true} : \mathtt{bool} \qquad \Gamma \vdash \mathsf{false} : \mathtt{bool} \qquad \Gamma, x : S \vdash x : S$$

$$\frac{\Gamma \vdash \mathsf{e} : \mathtt{nat}}{\Gamma \vdash \mathtt{succ}(\mathsf{e}) : \mathtt{nat}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{int}}{\Gamma \vdash \mathtt{neg}(\mathsf{e}) : \mathtt{int}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{bool}}{\Gamma \vdash \neg \mathsf{e} : \mathtt{bool}}$$

$$\frac{\Gamma \vdash \mathsf{e}_1 : S \quad \Gamma \vdash \mathsf{e}_2 : S}{\Gamma \vdash \mathsf{e}_1 \oplus \mathsf{e}_2 : S} \qquad \frac{\Gamma \vdash \mathsf{e}_1 : \mathtt{int} \quad \Gamma \vdash \mathsf{e}_2 : \mathtt{int}}{\Gamma \vdash \mathsf{e}_1 > \mathsf{e}_2 : \mathtt{bool}} \qquad \frac{\Gamma \vdash \mathsf{e} : S \quad S \leq : S'}{\Gamma \vdash \mathsf{e} : S'}$$

Table 7: Typing rules for expressions.

$$\frac{\forall i \in I \quad \Gamma, x : S_i \vdash P_i : \mathsf{T}_i}{\Gamma \vdash \sum_{i \in I} \mathsf{q}?\ell_i(x).P_i : \bigwedge_{i \in I} \mathsf{q}?\ell_i(S_i).\mathsf{T}_i}\ [\text{T-IN-CHOICE}] \qquad \Gamma \vdash \mathbf{0} : \mathtt{end}\ [\text{T-}\mathbf{0}]$$

$$\frac{\Gamma \vdash \mathsf{e} : S \quad \Gamma \vdash P : \mathsf{T}}{\Gamma \vdash \mathsf{q}!\ell(\mathsf{e}).P : \mathsf{q}!\ell(S).\mathsf{T}}\ [\text{T-OUT}] \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{bool} \quad \Gamma \vdash P_1 : \mathsf{T}_1 \quad \Gamma \vdash P_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 : \mathsf{T}_1 \vee \mathsf{T}_2}\ [\text{T-COND}]$$

$$\frac{\Gamma, X : \mathsf{T} \vdash P : \mathsf{T}}{\Gamma \vdash \mu X.P : \mathsf{T}}\ [\text{T-REC}] \quad \Gamma, X : \mathsf{T} \vdash X : \mathsf{T}\ [\text{T-VAR}] \quad \frac{\Gamma \vdash P : \mathsf{T} \qquad \mathsf{T} \leqslant \mathsf{T}'}{\Gamma \vdash P : \mathsf{T}'}\ [\text{T-SUB}]$$

$$\frac{\forall i \in I \quad \vdash P_i : \mathsf{G} \upharpoonright \mathsf{p}_i \quad \mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}_i \mid i \in I\}}{\vdash \prod_{i \in I} \mathsf{p}_i \triangleleft P_i : \mathsf{G}}\ [\text{T-SESS}]$$

Table 8: Typing rules for processes and sessions.

As usual we start with an inversion and a substitution lemmas.

Lemma 3.2 (Inversion lemma)

1. *Let* $\Gamma \vdash P : \mathsf{T}$.
   - *(a) If* $P = \sum_{i \in I} \mathsf{p}_i?\ell_i(x).Q_i$, *then* $\bigwedge_{i \in I} \mathsf{p}_i?\ell_i(S_i).\mathsf{T}_i \leqslant \mathsf{T}$ *and* $\Gamma, x : S_i \vdash Q_i : \mathsf{T}_i$.
   - *(b) If* $P = \mathsf{p}!\ell(\mathsf{e}).Q$, *then* $\mathsf{p}!\ell(S).\mathsf{T}' \leqslant \mathsf{T}$ *and* $\Gamma \vdash \mathsf{e} : S$ *and* $\Gamma \vdash Q : \mathsf{T}'$.
   - *(c) If* $P = \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$, *then* $\mathsf{T}_1 \vee \mathsf{T}_2 \leqslant \mathsf{T}$ *and* $\Gamma \vdash Q_1 : \mathsf{T}_1$ *and* $\Gamma \vdash Q_2 : \mathsf{T}_2$.
   - *(d) If* $P = \mu X.Q$, *then* $\Gamma, X : \mathsf{T} \vdash Q : \mathsf{T}$.
   - *(e) If* $P = X$, *then* $\Gamma = \Gamma', X : \mathsf{T}'$ *and* $\mathsf{T}' \leqslant \mathsf{T}$.
   - *(f) If* $P = \mathbf{0}$, *then* $\mathsf{T} = \mathtt{end}$.
2. *If* $\vdash \prod_{i \in I} \mathsf{p}_i \triangleleft P_i : \mathsf{G}$, *then* $\vdash P_i : \mathsf{G} \upharpoonright \mathsf{p}_i$ *for all* $i \in I$ *and* $\mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}_i \mid i \in I\}$.

Proof. By induction on type derivations.

Lemma 3.3 (Substitution lemma) *If* $\Gamma, x : S \vdash P : \mathsf{T}$ *and* $\Gamma \vdash \mathsf{v} : S$, *then* $\Gamma \vdash P\{\mathsf{v}/x\} : \mathsf{T}$.

Proof. By structural induction on P.

In order to state subject reduction we need to formalise how global types are modified by reducing multiparty sessions.

Definition 3.4 *1. The* consumption *of the communication* $\mathsf{p} \xrightarrow{\ell} \mathsf{q}$ *for the global type* G *(notation* $\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$*) is the global type inductively defined by:*

$$(\mathsf{r} \to \mathsf{s} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} = \begin{cases} \mathsf{G}_{i_0} & \text{if } \mathsf{r} = \mathsf{p}, \mathsf{s} = \mathsf{q}, \ell_{i_0} = \ell \\ \mathsf{r} \to \mathsf{s} : \{\ell_i(S_i).\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\}_{i\in I} & \text{otherwise} \end{cases}$$

$$(\mu \boldsymbol{t}.\mathsf{G}) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} = \mu \boldsymbol{t}.\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$$

2. The reduction of global types is the smallest pre-order relation closed under the rule:

$$\mathsf{G} \Longrightarrow \mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$$

Notice that $\mathtt{end} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ and $\mathbf{t} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ are undefined. It is easy to verify that, if G is projectable and $\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ is defined, then the global type $\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ is projectable. The following lemma shows other properties of consumption that are essential in the proof of subject reduction.

Lemma 3.5 *If* $\mathsf{q}!\ell(S).\mathsf{T} \leq \mathsf{G} \upharpoonright \mathsf{p}$ *and* $\mathsf{p}?\ell(S).\mathsf{T}' \wedge \mathsf{T}'' \leq \mathsf{G} \upharpoonright \mathsf{q}$*, then* $\mathsf{T} \leq (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{p}$ *and* $\mathsf{T}' \leq (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{q}$*. Moreover* $\mathsf{G} \upharpoonright \mathsf{r} = (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r}$ *for* $\mathsf{r} \neq \mathsf{p}$, $\mathsf{r} \neq \mathsf{q}$.

Proof. By induction on G and by cases on the definition of $\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$. Notice that G can only be $\mathsf{s}_1 \to \mathsf{s}_2 : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ with either $\mathsf{s}_1 = \mathsf{p}$ and $\mathsf{s}_2 = \mathsf{q}$ or $\{\mathsf{s}_1, \mathsf{s}_2\} \cap \{\mathsf{p}, \mathsf{q}\} = \emptyset$, since otherwise the types in the statement of the lemma could not be subtypes of the given projections of G.

If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$, then $\mathsf{G} \upharpoonright \mathsf{p} = \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{p}$ and $\mathsf{G} \upharpoonright \mathsf{q} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{q}$. From $\mathsf{q}!\ell(S).\mathsf{T} \leq \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{p}$ we get $\ell = \ell_{i_0}$ and $\mathsf{T} \leq \mathsf{G}_{i_0} \upharpoonright \mathsf{p}$ for some $i_0 \in I$. From $\mathsf{p}?\ell(S).\mathsf{T}' \wedge \mathsf{T}'' \leq \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{q}$ and $\ell = \ell_{i_0}$ we get $\mathsf{T}' \leq \mathsf{G}_{i_0} \upharpoonright \mathsf{q}$. We get $\mathsf{T} \leq (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{p}$ and $\mathsf{T}' \leq (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{q}$, since $(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{p} = \mathsf{G}_{i_0} \upharpoonright \mathsf{p}$ and $(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{q} = \mathsf{G}_{i_0} \upharpoonright \mathsf{q}$. If $\mathsf{r} \neq \mathsf{p}$, $\mathsf{r} \neq \mathsf{q}$, then by definition of projection $\mathsf{G} \upharpoonright \mathsf{r} = \mathsf{G}_{i_0} \upharpoonright \mathsf{r}$ for an arbitrary $i_0 \in I$, and then $\mathsf{G} \upharpoonright \mathsf{r} = (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r}$ by definition of consumption.

If $\mathsf{G} = \mathsf{s}_1 \to \mathsf{s}_2 : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ and $\{\mathsf{s}_1, \mathsf{s}_2\} \cap \{\mathsf{p}, \mathsf{q}\} = \emptyset$, then $\mathsf{G} \upharpoonright \mathsf{p} = \mathsf{G}_{i_0} \upharpoonright \mathsf{p}$ and $\mathsf{G} \upharpoonright \mathsf{q} = \mathsf{G}_{i_0} \upharpoonright \mathsf{q}$ for an arbitrary $i_0 \in I$. By definition of consumption

$$\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} = \mathsf{s}_1 \to \mathsf{s}_2 : \{\ell_i(S_i).\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\}_{i\in I},$$

which implies $(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{p} = (\mathsf{G}_{i_0} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{p}$ and $(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{q} = (\mathsf{G}_{i_0} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{q}$. Notice that the choice of i_0 does not modify the projection, by definition of projectability. We get $\mathsf{q}!\ell(S).\mathsf{T} \leq \mathsf{G}_{i_0} \upharpoonright \mathsf{p}$ and $\mathsf{p}?\ell(S).\mathsf{T}' \wedge \mathsf{T}'' \leq \mathsf{G}_{i_0} \upharpoonright \mathsf{q}$, which imply by induction $\mathsf{T} \leq (\mathsf{G}_{i_0} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{p}$ and $\mathsf{T}' \leq (\mathsf{G}_{i_0} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{q}$. If $\mathsf{r} = \mathsf{s}_1$, then $\mathsf{G} \upharpoonright \mathsf{r} = \bigvee_{i\in I} \mathsf{s}_2!\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{r}$ and

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r} = \bigvee_{i\in I} \mathsf{s}_2!\ell_i(S_i).(\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r},$$

so we conclude since by induction $\mathsf{G}_i \upharpoonright \mathsf{r} = (\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r}$ for all $i \in I$.
If $\mathsf{r} = \mathsf{s}_2$, then $\mathsf{G} \upharpoonright \mathsf{r} = \bigwedge_{i\in I} \mathsf{s}_1?\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{r}$ and

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r} = \bigwedge_{i\in I} \mathsf{s}_1?\ell_i(S_i).(\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r},$$

so we conclude since by induction $\mathsf{G}_i \upharpoonright \mathsf{r} = (\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r}$ for all $i \in I$.
If $\mathsf{r} \notin \{\mathsf{s}_1, \mathsf{s}_2\}$, then $\mathsf{G} \upharpoonright \mathsf{r} = \mathsf{G}_{i_0} \upharpoonright \mathsf{r}$ and $(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r} = (\mathsf{G}_{i_0} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r}$ for an arbitrary $i_0 \in I$. We can conclude using induction.

We can now prove subject reduction.

Theorem 3.6 (Subject reduction) *If* $\vdash \mathscr{M} : \mathsf{G}$ *and* $\mathscr{M} \longrightarrow^* \mathscr{M}'$*, then* $\vdash \mathscr{M}' : \mathsf{G}'$ *for some* G' *such that* $\mathsf{G} \Longrightarrow \mathsf{G}'$.

Proof. By induction on the multiparty session reduction. We only consider the case of rule [R-COMM] as premise of rule [R-CONTEXT]. In this case

$$\mathscr{M} \equiv \mathsf{p} \triangleleft \sum_{i\in I} \mathsf{q}?\ell_i(x).P_i \;\mid\; \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).P \mid \prod_{l\in L} \mathsf{p}_l \triangleleft Q_l$$

and

$$\mathscr{M}' \equiv \mathsf{p} \triangleleft P_j\{\mathsf{v}/x\} \;\mid\; \mathsf{q} \triangleleft P \mid \prod_{l\in L} \mathsf{p}_l \triangleleft Q_l,$$

where $j \in I$, $\mathsf{e} \downarrow \mathsf{v}$. By Lemma 3.2(2) $\vdash \mathscr{M} : \mathsf{G}$ implies $\vdash \sum_{i\in I} \mathsf{q}?\ell_i(x).P_i : \mathsf{G} \upharpoonright \mathsf{p}$, and $\vdash \mathsf{p}!\ell_j(\mathsf{e}).P : \mathsf{G} \upharpoonright \mathsf{q}$, and $\vdash Q_l : \mathsf{G} \upharpoonright \mathsf{p}_l$ for $l \in L$. By Lemma 3.2(1*a*) $\bigwedge_{i\in I} \mathsf{q}?\ell_i(S_i).\mathsf{T}_i \leqslant \mathsf{G} \upharpoonright \mathsf{p}$ and $x : S_i \vdash P_i : \mathsf{T}_i$ for $i \in I$. By Lemma 3.2(1*b*) $\mathsf{p}!\ell_j(S).\mathsf{T} \leqslant \mathsf{G} \upharpoonright \mathsf{q}$ and $\vdash \mathsf{e} : S$ and $\vdash P : \mathsf{T}$. From $\bigwedge_{i\in I} \mathsf{q}?\ell_i(S_i).\mathsf{T}_i \leqslant \mathsf{G} \upharpoonright \mathsf{p}$ and $\mathsf{p}!\ell_j(S).\mathsf{T} \leqslant \mathsf{G} \upharpoonright \mathsf{q}$ we get $S_j = S$. By Lemma 3.3 $x : S \vdash P_j : \mathsf{T}_j$ and $\vdash \mathsf{e} : S$ and $\mathsf{e} \downarrow \mathsf{v}$ imply $\vdash P_j\{\mathsf{v}/x\} : \mathsf{T}_j$. Then we choose $\mathsf{G}' = \mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_j} \mathsf{q}$, since Lemma 3.5 gives $\mathsf{T}_j \leq (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_j} \mathsf{q}) \upharpoonright \mathsf{p}$ and $\mathsf{T} \leq (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_j} \mathsf{q}) \upharpoonright \mathsf{q}$ and the same projections for all other participants of G.

To show progress a lemma on canonical forms is handy. The proof easily follows from the inspection of the typing rules.

Lemma 3.7 (Canonical forms)

1. If $\vdash P : \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i$, *then* $P = \sum_{i\in I'} \mathsf{p}?\ell_i(x).P_i$ *with* $I \subseteq I'$.

2. If $\vdash P : \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i$, *then* $\mathsf{q} \triangleleft P \longrightarrow^* \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).Q$ *with* $j \in I$.

Theorem 3.8 (Progress) *If* $\vdash \mathscr{M} : \mathsf{G}$, *then either* $\mathscr{M} \equiv \mathsf{p} \triangleleft \mathbf{0}$ *or* $\mathscr{M} \longrightarrow \mathscr{M}'$.

Proof. If $\mathsf{G} = \mathtt{end}$, then $\mathscr{M} \equiv \mathsf{p} \triangleleft \mathbf{0}$ by Lemma 3.2(2). If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$, then

$$\mathscr{M} \equiv \mathsf{p} \triangleleft P \mid \mathsf{q} \triangleleft Q \mid \mathscr{M}''$$

and $\vdash P : \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{p}$ and $\vdash Q : \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{G}_i \upharpoonright \mathsf{q}$ again by Lemma 3.2(2). By Lemma 3.7 $P = \sum_{i\in I'} \mathsf{p}?\ell_i(x).P_i$ with $I \subseteq I'$ and $\mathsf{q} \triangleleft Q \longrightarrow^* \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).Q'$ with $j \in I$. Therefore, if $\mathsf{e} \downarrow \mathsf{v}$, then $\mathscr{M} \longrightarrow^* \mathsf{p} \triangleleft P \mid \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).Q' \mid \mathscr{M}'' \longrightarrow \mathsf{p} \triangleleft P_j\{\mathsf{v}/x\} \mid \mathsf{q} \triangleleft Q' \mid \mathscr{M}''$.

The safety property that a typed multiparty session will never get stuck is a consequence of subject reduction and progress.

Theorem 3.9 (Safety) *If* $\vdash \mathscr{M} : \mathsf{G}$, *then it does not hold* $\mathtt{stuck}(\mathscr{M})$.

4 Operational Preciseness

We adapt the notion of operational preciseness [16, 3, 7] to our calculus.

Definition 4.1 *A subtyping relation is* operationally precise *if for any two types* T *and* T' *the following equivalence holds:*

$$\mathsf{T} \leqslant \mathsf{T}' \text{ if and only if there are no } P, \mathsf{p}, \mathscr{M} \text{ such that:}$$

- $\vdash P : \mathsf{T}$; *and*
- $\vdash Q : \mathsf{T}'$ *implies* $\vdash \mathsf{p} \triangleleft Q \mid \mathscr{M}$; *and*
- $\mathtt{stuck}(\mathsf{p} \triangleleft P \mid \mathscr{M})$.

The *operational soundness*, i.e. if for all Q such that $\vdash Q : \mathsf{T}'$ implies $\vdash \mathsf{p} \triangleleft Q \mid \mathscr{M}$, then $\mathsf{p} \triangleleft P \mid \mathscr{M}$ is not stuck, follows from the subsumption rule [T-SUB] and the safety theorem, Theorem 3.9.

To show the vice versa, it is handy to define the set $\mathtt{pt}\{\mathsf{T}\}$ of participants of a session type T as follows

$$\mathtt{pt}\{\textstyle\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i\} = \mathtt{pt}\{\textstyle\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i\} = \{\mathsf{p}\} \cup \textstyle\bigcup_{i\in I} \mathtt{pt}\{\mathsf{T}_i\}$$
$$\mathtt{pt}\{\mu\mathbf{t}.\mathsf{T}\} = \mathtt{pt}\{\mathsf{T}\} \qquad \mathtt{pt}\{\mathbf{t}\} = \mathtt{pt}\{\mathtt{end}\} = \emptyset$$

The proof of *operational completeness* comes in four steps.

$$\frac{\mathsf{T} \neq \mathtt{end}}{\mathsf{T} \not\preccurlyeq \mathtt{end}}\ [\text{NSUB-ENDL}] \qquad \frac{\mathsf{T} \neq \mathtt{end}}{\mathtt{end} \not\preccurlyeq \mathsf{T}}\ [\text{NSUB-ENDR}] \qquad \frac{\mathsf{p} \neq \mathsf{q} \qquad \dagger, \ddagger \in \{?, !\}}{\mathsf{p} \dagger \ell_1(S_1).\mathsf{T}_1 \not\preccurlyeq \mathsf{q} \ddagger \ell_2(S_2).\mathsf{T}_2}\ [\text{NSUB-DIFF-PART}]$$

$$\mathsf{p}!\ell_1(S_1).\mathsf{T}_1 \not\preccurlyeq \mathsf{p}?\ell_2(S_2).\mathsf{T}_2\ [\text{NSUB-OUT-IN}] \qquad \mathsf{p}?\ell_1(S_1).\mathsf{T}_1 \not\preccurlyeq \mathsf{p}!\ell_2(S_2).\mathsf{T}_2\ [\text{NSUB-IN-OUT}]$$

$$\frac{\ell_1 \neq \ell_2 \text{ or } S_2 \not\leq: S_1 \text{ or } \mathsf{T}_1 \not\preccurlyeq \mathsf{T}_2}{\mathsf{p}?\ell_1(S_1).\mathsf{T}_1 \not\preccurlyeq \mathsf{p}?\ell_2(S_2).\mathsf{T}_2}\ [\text{NSUB-IN-IN}] \qquad \frac{\ell_1 \neq \ell_2 \text{ or } S_1 \not\leq: S_2 \text{ or } \mathsf{T}_1 \not\preccurlyeq \mathsf{T}_2}{\mathsf{p}!\ell_1(S_1).\mathsf{T}_1 \not\preccurlyeq \mathsf{p}!\ell_2(S_2).\mathsf{T}_2}\ [\text{NSUB-OUT-OUT}]$$

$$\frac{\mathsf{T} \not\preccurlyeq \mathsf{T}_1 \text{ or } \mathsf{T} \not\preccurlyeq \mathsf{T}_2}{\mathsf{T} \not\preccurlyeq \mathsf{T}_1 \wedge \mathsf{T}_2}\ [\text{NSUB-INTR}] \qquad \frac{\mathsf{T}_1 \not\preccurlyeq \mathsf{T} \text{ or } \mathsf{T}_2 \not\preccurlyeq \mathsf{T}}{\mathsf{T}_1 \vee \mathsf{T}_2 \not\preccurlyeq \mathsf{T}}\ [\text{NSUB-UNIL}] \qquad \frac{\forall i \in I\ \forall j \in J\ \mathsf{T}_i \not\preccurlyeq \mathsf{T}'_j}{\bigwedge_{i \in I} \mathsf{T}_i \not\preccurlyeq \bigvee_{j \in J} \mathsf{T}'_j}\ [\text{NSUB-INTL-UNIR}]$$

Table 9: Negation of subtyping

- **[Step 1]** We characterise the negation of the subtyping relation by inductive rules (notation $\not\preccurlyeq$).
- **[Step 2]** For each type T and participant $\mathsf{p} \notin \mathtt{pt}\{\mathsf{T}\}$, we define a *characteristic global type* $\mathscr{G}(\mathsf{T}, \mathsf{p})$ such that $\mathscr{G}(\mathsf{T}, \mathsf{p}) \upharpoonright \mathsf{p} = \mathsf{T}$.
- **[Step 3]** For each type T, we define a *characteristic process* $\mathscr{P}(\mathsf{T})$ typed by T, which offers the series of interactions described by T.
- **[Step 4]** We prove that if $\mathsf{T} \not\preccurlyeq \mathsf{T}'$, then $\mathtt{stuck}(\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1 \leq i \leq n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i))$, where $\mathtt{pt}\{\mathsf{T}'\} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$, and $\mathsf{T}_i = \mathscr{G}(\mathsf{T}', \mathsf{p}) \upharpoonright \mathsf{p}_i$ for $1 \leq i \leq n$. Hence we achieve completeness by choosing $P = \mathscr{P}(\mathsf{T})$ and $\mathscr{M} = \prod_{1 \leq i \leq n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ in the definition of preciseness (Definition 4.1).

Negation of subtyping Table 9 gives the negation of subtyping, which uses the negation of subsorting $\not\leq:$ defined as expected. These rules say that a type different from `end` cannot be compared to `end`, two input or output types with different participants, or different labels, or with sorts or continuations which do not match, cannot be compared. The rules in the last line just take into account the set theoretic properties of intersection and union. One can show that either $\mathsf{T} \leqslant \mathsf{T}'$ or $\mathsf{T} \not\preccurlyeq \mathsf{T}'$ holds for two arbitrary types T, T'.

Lemma 4.2 *$\mathsf{T} \not\preccurlyeq \mathsf{T}'$ is the negation of $\mathsf{T} \leqslant \mathsf{T}'$.*

Proof. If $\mathsf{T} \not\preccurlyeq \mathsf{T}'$, then we can show $\mathsf{T} \not\leqslant \mathsf{T}'$ by induction on the derivation of $\mathsf{T} \not\preccurlyeq \mathsf{T}'$. We develop just two cases (the others are similar):

- base case [NSUB-DIFF-PART]. Then, $\mathsf{T} = \mathsf{p} \dagger \ell_1(S_1).\mathsf{T}_1$ and $\mathsf{T}' = \mathsf{q} \ddagger \ell_2(S_2).\mathsf{T}_2$ with $\mathsf{p} \neq \mathsf{q}$ and $\dagger, \ddagger \in \{?, !\}$. We can verify that T and T' do not match the conclusion of [SUB-END], nor [SUB-IN], nor [SUB-OUT] — hence, we conclude $\mathsf{T} \not\leqslant \mathsf{T}'$;
- inductive case [NSUB-INTL-UNIR]. Then, $\mathsf{T} = \bigwedge_{i \in I} \mathsf{T}_i$ and $\mathsf{T}' = \bigvee_{j \in J} \mathsf{T}'_j$; moreover, $\forall i \in I\ \forall j \in J$: $\mathsf{T}_i \not\preccurlyeq \mathsf{T}'_j$ — and thus, by the induction hypothesis, $\mathsf{T}_i \not\leqslant \mathsf{T}'_j$. We now notice that $\mathsf{T} \leqslant \mathsf{T}'$ could only possibly hold by rule [SUB-IN] when J is a singleton and by rule [SUB-OUT] when I is a singleton— but, since $\mathsf{T}_i \not\leqslant \mathsf{T}'_j$, at least one of the coinductive premises of such rules is not satisfied. Hence, we conclude $\mathsf{T} \not\leqslant \mathsf{T}'$.

$$\begin{array}{l}
\mathscr{G}_0(\bigwedge_{i\in I} \mathtt{p}_{j_0}?\ell_i(S_i).\mathsf{T}_i,\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n}) = \mathtt{p}_{j_0}\to\mathtt{p}:\{\ell_i(S_i).\mathsf{G}_i^{j_0}\}_{i\in I}\\
\mathscr{G}_0(\bigvee_{i\in I} \mathtt{p}_{j_0}!\ell_i(S_i).\mathsf{T}_i,\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n}) = \mathtt{p}\to\mathtt{p}_{j_0}:\{\ell_i(S_i).\mathsf{G}_i^{j_0}\}_{i\in I}\\
\mathscr{G}_0(\mu\mathbf{t}.\mathsf{T},\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n}) = \mu\mathbf{t}.\mathscr{G}_0(\mathsf{T},\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n})\\
\mathscr{G}_0(\mathbf{t},\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n}) = \mathbf{t} \qquad \mathscr{G}_0(\mathtt{end},\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n}) = \mathtt{end}\\
\mathsf{G}_i^{j_0} \;=\; \mathtt{p}_{j_0}\to\mathtt{p}_{j_0+1}:\ell_i(\mathtt{bool}).\ldots\mathtt{p}_{n-1}\to\mathtt{p}_n:\ell_i(\mathtt{bool}).\mathtt{p}_n\to\mathtt{p}_1:\ell_i(\mathtt{bool}).\\
\qquad\quad \mathtt{p}_1\to\mathtt{p}_2:\ell_i(\mathtt{bool}).\ldots.\mathtt{p}_{j_0-1}\to\mathtt{p}_{j_0}:\ell_i(\mathtt{bool}).\mathscr{G}_0(\mathsf{T}_i,\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n})
\end{array}$$

Table 10: The function $\mathscr{G}_0(\mathsf{T},\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n})$.

Vice versa, assume $\mathsf{T}\not\leqslant\mathsf{T}'$: if we try to apply the subtyping rules to show $\mathsf{T}\leqslant\mathsf{T}'$, we will "fail" after n derivation steps, by finding two types $\mathsf{T}_1,\mathsf{T}_2$ whose syntactic shapes do *not* match the conclusion of [SUB-END], nor [SUB-IN], nor [SUB-OUT]. We prove $\mathsf{T}\not\trianglelefteq\mathsf{T}'$ by induction on n:

- base case $n=0$. The derivation "fails" immediately, i.e. $\mathsf{T}_1=\mathsf{T}$ and $\mathsf{T}_2=\mathsf{T}'$. By cases on the possible shapes of T and T', we obtain $\mathsf{T}\not\trianglelefteq\mathsf{T}'$ by one of the rules [NSUB-ENDL], [NSUB-ENDR], [NSUB-DIFF-PART], [NSUB-OUT-IN], [NSUB-IN-OUT], [NSUB-IN-IN], [NSUB-OUT-OUT];
- inductive case $n=m+1$. The shapes of T,T' match the conclusion of [SUB-IN] (resp. [SUB-OUT]), but there is some coinductive premise $\mathsf{T}_1\leqslant\mathsf{T}_2$ whose sub-derivation "fails" after m steps. By the induction hypothesis, we have $\mathsf{T}_1\not\trianglelefteq\mathsf{T}_2$: therefore, we can derive $\mathsf{T}\not\trianglelefteq\mathsf{T}'$ by one of the rules [NSUB-IN-IN] or [NSUB-INTR] (or [NSUB-OUT-OUT] or [NSUB-UNIL]) or [NSUB-INTL-UNIR].

Characteristic global types The characteristic global type $\mathscr{G}(\mathsf{T},\mathtt{p})$ of the type T for the participant $\mathtt{p}$ describes the communications between $\mathtt{p}$ and all participants in $\mathtt{pt}\{\mathsf{T}\}$ following T. In fact after each communication involving $\mathtt{p}$ and some $\mathtt{q}\in\mathtt{pt}\{\mathsf{T}\}$, $\mathtt{q}$ starts a cyclic communication involving all participants in $\mathtt{pt}\{\mathsf{T}\}$ both as receivers and senders. This is needed for getting both a projectable global type and a stuck session, see the proof of Theorem 4.4 and Examples 4.3 and 4.5. More precisely, we define the characteristic global type $\mathscr{G}(\mathsf{T},\mathtt{p})$ of the type T for the participant $\mathtt{p}\notin\mathtt{pt}\{\mathsf{T}\}$ as $\mathscr{G}(\mathsf{T},\mathtt{p})=\mathscr{G}_0(\mathsf{T},\mathtt{p},\mathtt{pt}\{\mathsf{T}\})$, where $\mathscr{G}_0(\mathsf{T},\mathtt{p},\{\mathtt{p}_j\}_{1\le j\le n})$ is given in Table 10.

Example 4.3 *Some characteristic global types are projectable thanks to the cyclic communication. Take for example* $\mathsf{T}=\mathtt{q}!\ell_1(\mathtt{nat}).\mathtt{r}?\ell_2(\mathtt{int}).\mathtt{end}\vee\mathtt{q}!\ell_3(\mathtt{int}).\mathtt{end}$. *Without the cyclic communication we would get the global type* $\mathsf{G}=\mathtt{p}\to\mathtt{q}:\{\ell_1(\mathtt{nat}).\mathtt{r}\to\mathtt{p}:\ell_2(\mathtt{int}).\mathtt{end},\ell_3(\mathtt{int}).\mathtt{end}\}$ *and* $\mathsf{G}\upharpoonright\mathtt{r}=\mathtt{p}!\ell_2(\mathtt{int}).\mathtt{end}\between\mathtt{end}$ *is undefined. Instead*

$$\begin{array}{l}
\mathscr{G}(\mathsf{T},\mathtt{p}) = \mathtt{p}\to\mathtt{q}:\{\ell_1(\mathtt{nat}).\mathtt{q}\to\mathtt{r}:\ell_1(\mathtt{bool}).\mathtt{r}\to\mathtt{q}:\ell_1(\mathtt{bool}).\\
\qquad\qquad\qquad \mathtt{r}\to\mathtt{p}:\ell_2(\mathtt{int}).\mathtt{r}\to\mathtt{q}:\ell_2(\mathtt{bool}).\mathtt{q}\to\mathtt{r}:\ell_2(\mathtt{bool}).\mathtt{end},\\
\qquad\qquad\qquad \ell_3(\mathtt{int}).\mathtt{q}\to\mathtt{r}:\ell_3(\mathtt{bool}).\mathtt{r}\to\mathtt{q}:\ell_3(\mathtt{bool}).\mathtt{end}\}\\
\mathscr{G}(\mathsf{T},\mathtt{p})\upharpoonright\mathtt{r} = \mathtt{q}?\ell_1(\mathtt{bool}).\mathtt{q}!\ell_1(\mathtt{bool}).\mathtt{p}!\ell_2(\mathtt{int}).\mathtt{q}!\ell_2(\mathtt{bool}).\mathtt{q}?\ell_2(\mathtt{bool}).\mathtt{end}\between\\
\qquad\qquad \mathtt{q}?\ell_3(\mathtt{bool}).\mathtt{q}!\ell_3(\mathtt{bool}).\mathtt{end}
\end{array}$$

It is easy to verify that $\mathscr{G}(\mathsf{T},\mathtt{p})\upharpoonright\mathtt{p}=\mathsf{T}$ and $\mathscr{G}(\mathsf{T},\mathtt{p})\upharpoonright\mathtt{q}$ is defined for all $\mathtt{q}\in\mathtt{pt}\{\mathsf{T}\}$ by induction on the definition of characteristic global types.

Characteristic processes We define the characteristic process $\mathscr{P}(\mathsf{T})$ of the type T by using the operators succ, neg, and $\neg$ to check if the received values are of the right sort and exploiting the correspondence between external choices and intersections, conditionals and unions. Conditionals also allow the evaluation of expressions which can be stuck. The definition of $\mathscr{P}(\mathsf{T})$ by induction on T is given in Table 11. By induction on the structure of $\mathscr{P}(\mathsf{T})$ it is easy to verify that $\vdash\mathscr{P}(\mathsf{T}):\mathsf{T}$.

We have now all the necessary machinery to show operational preciseness of subtyping.

$$\mathscr{P}(\mathsf{T}) = \begin{cases} \mathsf{p}?\ell(x).\text{if } \mathtt{succ}(x) > 0 \text{ then } \mathscr{P}(\mathsf{T}') \text{ else } \mathscr{P}(\mathsf{T}') & \text{if } \mathsf{T} = \mathsf{p}?\ell(\mathtt{nat}).\mathsf{T}', \\ \mathsf{p}?\ell(x).\text{if } \mathtt{neg}(x) > 0 \text{ then } \mathscr{P}(\mathsf{T}') \text{ else } \mathscr{P}(\mathsf{T}') & \text{if } \mathsf{T} = \mathsf{p}?\ell(\mathtt{int}).\mathsf{T}', \\ \mathsf{p}?\ell(x).\text{if } \neg x \text{ then } \mathscr{P}(\mathsf{T}') \text{ else } \mathscr{P}(\mathsf{T}') & \text{if } \mathsf{T} = \mathsf{p}?\ell(\mathtt{bool}).\mathsf{T}', \\ \mathsf{p}!\ell(5).\mathscr{P}(\mathsf{T}') & \text{if } \mathsf{T} = \mathsf{p}!\ell(\mathtt{nat}).\mathsf{T}', \\ \mathsf{p}!\ell(-5).\mathscr{P}(\mathsf{T}') & \text{if } \mathsf{T} = \mathsf{p}!\ell(\mathtt{int}).\mathsf{T}', \\ \mathsf{p}!\ell(\mathsf{true}).\mathscr{P}(\mathsf{T}') & \text{if } \mathsf{T} = \mathsf{p}!\ell(\mathtt{bool}).\mathsf{T}', \\ \mathscr{P}(\mathsf{T}_1) + \mathscr{P}(\mathsf{T}_2) & \text{if } \mathsf{T} = \mathsf{T}_1 \wedge \mathsf{T}_2, \\ \text{if } \mathsf{true} \oplus \mathsf{false} \text{ then } \mathscr{P}(\mathsf{T}_1) \text{ else } \mathscr{P}(\mathsf{T}_2) & \text{if } \mathsf{T} = \mathsf{T}_1 \vee \mathsf{T}_2, \\ \mu X_{\mathbf{t}}.\mathscr{P}(\mathsf{T}') & \text{if } \mathsf{T} = \mu\mathbf{t}.\mathsf{T}', \\ X_{\mathbf{t}} & \text{if } \mathsf{T} = \mathbf{t}, \\ \mathbf{0} & \text{if } \mathsf{T} = \mathtt{end}. \end{cases}$$

Table 11: Characteristic processes

Theorem 4.4 (Preciseness) *The synchronous multiparty session subtyping is operationally precise.*

Proof. We only need to show completeness of the synchronous multiparty session subtyping. Let $\mathsf{T} \leqslant \mathsf{T}'$ and $\mathsf{p} \notin \mathtt{pt}\{\mathsf{T}'\} = \{\mathsf{p}_i\}_{1\leq i\leq n}$ and $\mathsf{G} = \mathscr{G}(\mathsf{T}', \mathsf{p})$ and $\mathsf{T}_i = \mathsf{G} \upharpoonright \mathsf{p}_i$ for $1 \leq i \leq n$. Then $\vdash Q : \mathsf{T}'$ implies $\vdash \mathsf{p} \triangleleft Q \mid \prod_{1\leq i\leq n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ by rule [T-SESS]. We show that

$$\mathtt{stuck}(\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1\leq i\leq n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)).$$

The proof is by induction on the definition of $\not\trianglelefteq$. We only consider some interesting cases.

[NSUB-DIFF-PART]

$$\frac{\mathsf{q} \neq \mathsf{p}_h \qquad \dagger, \ddagger \in \{?, !\}}{\mathsf{q} \dagger \ell(S).\mathsf{T}_0 \not\trianglelefteq \mathsf{p}_h \ddagger \ell'(S').\mathsf{T}_0'}$$

By definition $\mathscr{P}(\mathsf{T}) = \mathsf{q}\dagger\ell(\mathsf{e}).P$ for suitable e, P. If $\mathsf{q} \notin \{\mathsf{p}_i\}_{1\leq i\leq n}$, then

$$\mathtt{stuck}(\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1\leq i\leq n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)),$$

since $\mathscr{P}(\mathsf{T})$ will never communicate.

Otherwise let $\mathsf{q} = \mathsf{p}_j$ with $1 \leq j \leq n$ and $j \neq h$. By construction $\mathscr{P}(\mathsf{T}_h) = \mathsf{p}\overline{\ddagger}\ell'(\mathsf{e}_h).P_h$, where $\overline{\ddagger} = \begin{cases} ? & \text{if } \ddagger = ! \\ ! & \text{if } \ddagger = ? \end{cases}$, and $\mathscr{P}(\mathsf{T}_k) = \mathsf{p}_{f(k)}?\ell'(x).P_k$, where $f(k) = \begin{cases} k-1 & \text{if } k > 1 \\ n & \text{if } k = 1 \end{cases}$ for $1 \leq k \leq n$ and $k \neq h$.

Therefore $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1\leq i\leq n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ cannot reduce.

[NSUB-IN-IN]

$$\frac{\ell_1 \neq \ell_2 \text{ or } S_2 \not\leqslant : S_1 \text{ or } \mathsf{T}_1 \not\trianglelefteq \mathsf{T}_2}{\mathsf{p}_h?\ell_1(S_1).\mathsf{T}_1 \not\trianglelefteq \mathsf{p}_h?\ell_2(S_2).\mathsf{T}_2}$$

A paradigmatic case is $\ell_1 = \ell_2 = \ell$, $S_1 = \mathtt{nat}$, $S_2 = \mathtt{int}$, $\mathsf{T}_1 = \mathsf{T}_2 = \mathtt{end}$. By definition $\mathtt{pt}\{\mathsf{T}'\} = \{\mathsf{p}_h\}$ and $\mathscr{P}(\mathsf{T}) = \mathsf{p}_h?\ell(x).\text{if } \mathtt{succ}(x) > 0 \text{ then } \mathbf{0} \text{ else } \mathbf{0}$ and $\mathscr{P}(\mathsf{T}_h) = \mathsf{p}!\ell(-5).\mathbf{0}$. Therefore $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \mathscr{P}(\mathsf{T}_h)$ reduces to $\mathsf{p} \triangleleft \text{if } \mathtt{succ}(-5) > 0 \text{ then } \mathbf{0} \text{ else } \mathbf{0}$, which is stuck.

[NSUB-INTR]

$$\frac{\mathsf{T} \not\trianglelefteq \mathsf{T}_1' \text{ or } \mathsf{T} \not\trianglelefteq \mathsf{T}_2'}{\mathsf{T} \not\trianglelefteq \mathsf{T}_1' \wedge \mathsf{T}_2'}$$

By definition T_1' and T_2' must be intersections of inputs with the same sender, let it be p_h. Let $\mathsf{G}_1 = \mathscr{G}(\mathsf{T}_1', \mathsf{p})$, $\mathsf{G}_2 = \mathscr{G}(\mathsf{T}_2', \mathsf{p})$, $P_h^{(1)} = \mathscr{P}(\mathsf{G}_1 \upharpoonright \mathsf{p}_h)$, $P_h^{(2)} = \mathscr{P}(\mathsf{G}_2 \upharpoonright \mathsf{p}_h)$. Then by construction

$$P_h = \mathscr{P}(\mathscr{G}(\mathsf{T}'_1 \wedge \mathsf{T}'_2, \mathsf{p}) \upharpoonright \mathsf{p}_h) = \mathsf{if}\ \mathsf{true} \oplus \mathsf{false}\ \mathsf{then}\ P_h^{(1)}\ \mathsf{else}\ P_h^{(2)}.$$

This implies that $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ reduces to both $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \mathsf{p}_h \triangleleft P_h^{(1)} \mid \prod_{1\le i\ne h\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ and $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \mathsf{p}_h \triangleleft P_h^{(2)} \mid \prod_{1\le i\ne h\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$. By induction either $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \mathsf{p}_h \triangleleft P_h^{(1)} \mid \prod_{1\le i\ne h\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ or $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \mathsf{p}_h \triangleleft P_h^{(2)} \mid \prod_{1\le i\ne h\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ is stuck, and therefore also $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ is stuck.

[NSUB-UNIL]
$$\frac{\mathsf{T}'_1 \not\trianglelefteq \mathsf{T} \text{ or } \mathsf{T}'_2 \not\trianglelefteq \mathsf{T}}{\mathsf{T}'_1 \vee \mathsf{T}'_2 \not\trianglelefteq \mathsf{T}}$$

By definition T'_1 and T'_2 must be unions of outputs with the same receiver, let it be p_h. By definition $\mathscr{P}(\mathsf{T}'_1 \vee \mathsf{T}'_2) = \mathsf{if}\ \mathsf{true} \oplus \mathsf{false}\ \mathsf{then}\ \mathscr{P}(\mathsf{T}'_1)\ \mathsf{else}\ \mathscr{P}(\mathsf{T}'_2)$. Then $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}'_1 \vee \mathsf{T}'_2) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ reduces to both $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}'_1) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ and $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}'_2) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$. By induction

$$\text{either } \mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}'_1) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i) \text{ or } \mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}'_2) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i) \text{ is stuck,}$$

and therefore $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}'_1 \vee \mathsf{T}'_2) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ is stuck too.

[NSUB-INTL-UNIR]
$$\frac{\forall l \in L\ \forall j \in J\ \mathsf{T}'_l \not\trianglelefteq \mathsf{T}''_j}{\bigwedge_{l\in L} \mathsf{T}'_l \not\trianglelefteq \bigvee_{j\in J} \mathsf{T}''_j}$$

If L and J are both singleton sets it is immediate by induction.
If L and J both contain more than one index, then by definition we can assume (without loss of generality) that T'_l for $l \in L$ are input types with the same sender, let it be p_h, and T''_j for $j \in J$ are output types with the same receiver, let it be p_k. By definition $\mathscr{P}(\mathsf{T}) = \sum_{l\in L} \mathsf{p}_h?\ell_l(x).P'_l$, and $\mathscr{P}(\mathsf{T}_k) = \sum_{j\in J} \mathsf{p}?\ell_j(x).P''_j$ and $\mathscr{P}(\mathsf{T}_u) = \mathsf{p}_{f(u)}?\ell_j(x).P_u$, where f is as in the case of rule [NSUB-DIFF-PART], for $1 \le u \le n$ and $u \ne k$. Therefore $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ cannot reduce.
Let L contains more than one index and J be a singleton set. By definition $\mathscr{P}(\mathsf{T}) = \sum_{l\in L} P'_l$, where $P'_l = \mathscr{P}(\mathsf{T}'_l)$ for $l \in L$. Let us assume ad absurdum that $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ is not stuck. Then there must be $l_0 \in L$ such that $\mathsf{p} \triangleleft P'_{l_0} \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ is not stuck, contradicting the hypothesis.
If L is a singleton set and J contains more than one index, then T''_j for $j \in J$ must be unions of outputs with the same receiver, let it be p_h. Let $\mathsf{G}_j = \mathscr{G}(\mathsf{T}''_j, \mathsf{p})$ and $P_h^{(j)} = \mathscr{P}(\mathsf{G}_j \upharpoonright \mathsf{p}_h)$. Then $P_h = \mathscr{P}(\mathscr{G}(\bigvee_{j\in J} \mathsf{T}''_j, \mathsf{p}) \upharpoonright \mathsf{p}_h) = \sum_{j\in J} P_h^{(j)}$. Let us assume ad absurdum that $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \prod_{1\le i\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ is not stuck. In this case there must be $j_0 \in J$ such that $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \mathsf{p}_h \triangleleft P_h^{(j_0)} \mid \prod_{1\le i\ne h\le n} \mathsf{p}_i \triangleleft \mathscr{P}(\mathsf{T}_i)$ is not stuck, contradicting the hypothesis.

Example 4.5 *An example showing the utility of the cyclic communication in the definition of characteristic global types is* $\mathsf{T} = \mathsf{p}_1!\ell_1(\mathtt{nat}).\mathsf{p}_2!\ell_2(\mathtt{nat}).\mathtt{end}$ *and* $\mathsf{T}' = \mathsf{p}_2!\ell_2(\mathtt{nat}).\mathsf{p}_1!\ell_1(\mathtt{nat}).\mathtt{end}$. *In fact without the cyclic communication the characteristic global type of* T' *would be*

$$\mathsf{G} = \mathsf{p} \to \mathsf{p}_2 : \ell_2(\mathtt{nat}).\mathsf{p} \to \mathsf{p}_1 : \ell_1(\mathtt{nat}).\mathtt{end}$$

and then $\mathscr{M} = \mathtt{p}_1 \triangleleft \mathscr{P}(\mathsf{G} \upharpoonright \mathtt{p}_1) \mid \mathtt{p}_2 \triangleleft \mathscr{P}(\mathsf{G} \upharpoonright \mathtt{p}_2) = \mathtt{p}_1 \triangleleft \mathtt{p}?\ell_1(x).\mathbf{0} \mid \mathtt{p}_2 \triangleleft \mathtt{p}?\ell_2(x).\mathbf{0}$. *Being* $\mathscr{P}(\mathsf{T}) = \mathtt{p}_1!\ell_1(5).\mathtt{p}_2!\ell_2(5).\mathbf{0}$, *the session* $\mathtt{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \mathscr{M}$ *reduces to* $\mathtt{p} \triangleleft \mathbf{0}$. *Instead*

$$\begin{array}{rcl} \mathscr{G}(\mathsf{T}', \mathtt{p}) & = & \mathtt{p} \to \mathtt{p}_2 : \ell_2(\mathtt{nat}).\mathtt{p}_2 \to \mathtt{p}_1 : \ell_2(\mathtt{bool}).\mathtt{p}_1 \to \mathtt{p}_2 : \ell_2(\mathtt{bool}). \\ & & \mathtt{p} \to \mathtt{p}_1 : \ell_1(\mathtt{nat}).\mathtt{p}_1 \to \mathtt{p}_2 : \ell_1(\mathtt{bool}).\mathtt{p}_2 \to \mathtt{p}_1 : \ell_1(\mathtt{bool}).\mathtt{end}, \end{array}$$

which implies $\mathscr{P}(\mathscr{G}(\mathsf{T}', \mathtt{p}) \upharpoonright \mathtt{p}_1) = \mathtt{p}_2?\ell_2(x)\ldots$. *and* $\mathscr{P}(\mathscr{G}(\mathsf{T}', \mathtt{p}) \upharpoonright \mathtt{p}_2) = \mathtt{p}?\ell_2(x)\ldots$. *It is then easy to verify that* $\mathtt{p} \triangleleft \mathscr{P}(\mathsf{T}) \mid \mathtt{p}_1 \triangleleft \mathscr{P}(\mathscr{G}(\mathsf{T}', \mathtt{p}) \upharpoonright \mathtt{p}_1) \mid \mathtt{p}_2 \triangleleft \mathscr{P}(\mathscr{G}(\mathsf{T}', \mathtt{p}) \upharpoonright \mathtt{p}_2)$ *is stuck.*

5 Operational Preciseness at Work

Consider a multiparty session with four participants: client ($\mathtt{cl}$), adder ($\mathtt{add}$), increment ($\mathtt{inc}$), and decrement ($\mathtt{dec}$)

$$\mathtt{cl} \triangleleft \mathtt{P}_{\mathtt{cl}} \mid \mathtt{add} \triangleleft \mathtt{P}_{\mathtt{add}} \mid \mathtt{inc} \triangleleft \mathtt{P}_{\mathtt{inc}} \mid \mathtt{dec} \triangleleft \mathtt{P}_{\mathtt{dec}}.$$

Client sends two natural numbers to adder and expects the integer result of summation. Adder receives the two numbers and sum them by successively increasing the first one by 1 (done by $\mathtt{inc}$) and decreasing the second one by 1 (done by $\mathtt{dec}$). If the second summand equals 0, the first summand gives the required sum. Processes modelling this behaviour are the following:

$$\begin{array}{rcl} P_{\mathtt{cl}} & = & \mathtt{add}!\ell_1(5).\mathtt{add}!\ell_2(4).\mathtt{add}?\ell_3(\mathtt{x}).\mathbf{0} \\ P_{\mathtt{add}} & = & \mathtt{cl}?\ell_1(\mathtt{y}_1).\mathtt{cl}?\ell_2(\mathtt{y}_2).\mu\mathtt{X}.\mathtt{if}\ \mathtt{y}_2 = 0\ \mathtt{then}\ \mathtt{inc}!\ell_4(\mathtt{true}).\mathtt{dec}!\ell_4(\mathtt{true}).\mathtt{cl}!\ell_3(\mathtt{y}_1).\mathtt{end} \\ & & \qquad\qquad \mathtt{else}\ \mathtt{inc}!\ell_5(\mathtt{y}_1).\mathtt{inc}?\ell_6(\mathtt{y}_1).\mathtt{dec}!\ell_7(\mathtt{y}_2).\mathtt{dec}?\ell_8(\mathtt{y}_2).\mathtt{X} \\ P_{\mathtt{inc}} & = & \mu X.\mathtt{add}?\ell_4(\mathtt{bool}).\mathtt{end} + \mathtt{add}?\ell_5(\mathtt{y}).\mathtt{add}!\ell_6(\mathtt{y}+1).\mathtt{X} \\ P_{\mathtt{dec}} & = & \mu X.\mathtt{add}?\ell_4(\mathtt{bool}).\mathtt{end} + \mathtt{add}?\ell_7(\mathtt{y}).\mathtt{add}!\ell_8(\mathtt{y}-1).\mathtt{X}. \end{array}$$

We can extend addition to integers by changing the process $P_{\mathtt{add}}$ as follows:

$$\begin{array}{rcl} P'_{\mathtt{add}} & = & \mathtt{cl}?\ell_1(\mathtt{y}_1).\mathtt{cl}?\ell_2(\mathtt{y}_2).\mu\mathtt{X}.\mathtt{if}\ \mathtt{y}_2 = 0\ \mathtt{then}\ \mathtt{inc}!\ell_4(\mathtt{true}).\mathtt{dec}!\ell_4(\mathtt{true}).\mathtt{cl}!\ell_3(\mathtt{y}_1).\mathtt{end} \\ & & \qquad\qquad \mathtt{else\ if}\ y_2 > 0\ \mathtt{then}\ \mathtt{inc}!\ell_5(\mathtt{y}_1).\mathtt{inc}?\ell_6(\mathtt{y}_1).\mathtt{dec}!\ell_7(\mathtt{y}_2).\mathtt{dec}?\ell_8(\mathtt{y}_2).\mathtt{X} \\ & & \qquad\qquad\qquad \mathtt{else}\ \mathtt{inc}!\ell_5(\mathtt{y}_2).\mathtt{inc}?\ell_6(\mathtt{y}_2).\mathtt{dec}!\ell_7(\mathtt{y}_1).\mathtt{dec}?\ell_8(\mathtt{y}_1).\mathtt{X}. \end{array}$$

Process $P'_{\mathtt{add}}$ additionally checks if the second summand is positive. If it is not, the sum is calculated by successively increasing the second summand by 1 and decreasing the first summand by 1. The new multiparty session follows the global protocol

$$\begin{array}{l} \mathtt{cl} \to \mathtt{add} : \ell_1(\mathtt{int}).\mathtt{cl} \to \mathtt{add} : \ell_2(\mathtt{int}).\mu\mathbf{t}.\mathtt{add} \to \mathtt{inc} : \{ \\ \ell_4(\mathtt{bool}) : \mathtt{add} \to \mathtt{dec} : \ell_4(\mathtt{bool}).\mathtt{add} \to \mathtt{cl} : \ell_3(\mathtt{int}).\mathtt{end}, \\ \ell_5(\mathtt{int}).\mathtt{inc} \to \mathtt{add} : \ell_6(\mathtt{int}).\mathtt{add} \to \mathtt{dec} : \ell_7(\mathtt{int}).\mathtt{dec} \to \mathtt{add} : \ell_8(\mathtt{int}).\mathbf{t}\}. \end{array}$$

Operational soundness of the subtyping guarantees that the summation of natural numbers will be safe after this change, as for $\mathtt{nat} \leqslant \mathtt{int}$ we have

$$\mathtt{add}!\ell_1(\mathtt{nat}).\mathtt{add}!\ell_2(\mathtt{nat}).\mathtt{add}?\ell_3(\mathtt{int}).\mathtt{end} \leqslant \mathtt{add}!\ell_1(\mathtt{int}).\mathtt{add}!\ell_2(\mathtt{int}).\mathtt{add}?\ell_3(\mathtt{int}).\mathtt{end}.$$

On the other hand, by operational completeness we cannot swap sending of messages with different labels, e.g.

$$\mathsf{T} = \mathtt{add}!\ell_1(\mathtt{int}).\mathtt{add}!\ell_2(\mathtt{int}).\mathtt{end} \not\leqslant \mathtt{add}!\ell_2(\mathtt{int}).\mathtt{add}!\ell_1(\mathtt{int}).\mathtt{end} = \mathsf{T}'.$$

We can construct processes $Q_{\mathtt{cl}} = \mathtt{add}!\ell_1(5).\mathtt{add}!\ell_2(4).\mathbf{0}$ of type T and $Q'_{\mathtt{cl}} = \mathtt{add}!\ell_2(4).\mathtt{add}!\ell_1(5).\mathbf{0}$ of type T' and a multiparty session

$$\mathscr{M} = \mathtt{add} \triangleleft \mathtt{cl}?\ell_2(\mathtt{x}).\mathtt{if}\ \mathtt{neg}(\mathtt{x}) > 0\ \mathtt{then}\ \mathtt{cl}?\ell_1(\mathtt{x}).\mathbf{0}\ \mathtt{else}\ \mathtt{cl}?\ell_1(\mathtt{x}).\mathbf{0}$$

such that $\mathtt{cl} \triangleleft Q'_{\mathtt{cl}} \mid \mathscr{M}$ is well typed, while $\mathtt{cl} \triangleleft Q_{\mathtt{cl}} \mid \mathscr{M}$ is stuck, since the multiparty session

$\mathtt{cl} \lhd \mathtt{add}!\ell_1(5).\mathtt{add}!\ell_2(4).\mathbf{0} \mid \mathtt{add} \lhd \mathtt{cl}?\ell_2(\mathtt{x}).\mathsf{if}\ \mathtt{neg}(\mathtt{x}) > 0\ \mathsf{then}\ \mathtt{cl}?\ell_1(\mathtt{x}).\mathbf{0}\ \mathsf{else}\ \mathtt{cl}?\ell_1(\mathtt{x}).\mathbf{0}.$

cannot reduce because of label mismatch.

6 Denotational Preciseness

In λ-calculus types are usually interpreted as subsets of the domains of λ-models [1, 11]. *Denotational preciseness* of subtyping is then:

$$\mathsf{T} \leqslant \mathsf{T}' \text{ if and only if } [\![\mathsf{T}]\!] \subseteq [\![\mathsf{T}']\!],$$

using $[\![\ ]\!]$ to denote type interpretation.

In the present context let us interpret a session type T as the set of closed processes typed by T, i.e.

$$[\![\mathsf{T}]\!] = \{P \mid \vdash P : \mathsf{T}\}$$

We can then show that the subtyping is denotationally precise. The subsumption rule [T-SUB] gives the denotational soundness. Denotational completeness follows from the following key property of characteristic processes:

$$\vdash \mathscr{P}(\mathsf{T}) : \mathsf{T}' \text{ implies } \mathsf{T} \leqslant \mathsf{T}'.$$

If we could derive $\vdash \mathscr{P}(\mathsf{T}) : \mathsf{T}'$ with $\mathsf{T} \not\leqslant \mathsf{T}'$, then the multiparty session

$$\mathsf{p} \lhd \mathscr{P}(\mathsf{T}) \mid \prod_{1 \le i \le n} \mathsf{p}_i \lhd \mathscr{P}(\mathsf{T}_i),$$

where $\mathtt{pt}\{\mathsf{T}'\} = \{\mathsf{p}_i\}_{1 \le i \le n}$ and $\mathsf{G} = \mathscr{G}(\mathsf{T}', \mathsf{p})$ and $\mathsf{T}_i = \mathsf{G} \upharpoonright \mathsf{p}_i$ for $1 \le i \le n$, could be typed. Theorem 4.4 shows that this process is stuck, and this contradicts the soundness of the type system. We get the desired property, which implies denotational completeness, since if $\mathsf{T} \not\leqslant \mathsf{T}'$, then $\mathscr{P}(\mathsf{T}) \in [\![\mathsf{T}]\!]$, but $\mathscr{P}(\mathsf{T}) \notin [\![\mathsf{T}']\!]$.

Theorem 6.1 (Denotational preciseness) *The subtyping relations is denotationally precise.*

7 Conclusion

The preciseness result of this paper shows a rigorousness of the subtyping, which is implemented (as a default) in most of session-based programming languages and tools [14, 5, 12, 10] for enlarging typability.

The main technical contribution is the definition of characteristic global types, see Section 4. Given a session type T and a session participant p which does not occur in T, the associated characteristic global type expresses the communications prescribed by T between p and the participants in T. After each communication involving p, the characteristic global type creates a cyclic communication between all participants in T. Such a cyclic communication is essential to project the characteristic global type and to generate deadlock when the the subtyping relation is extended.

The subtyping considered here is sound but not complete for asynchronous multiparty sessions [13], as shown in [17]. We conjecture the completeness of the subtyping defined in [17] for asynchronous multiparty sessions and we are working toward this proof.

Acknowledgments. We are grateful to the anonymous reviewers for their useful remarks.

References

[1] Henk Barendregt, Mario Coppo & Mariangiola Dezani-Ciancaglini (1983): *A Filter Lambda Model and the Completeness of Type Assignment*. *Journal of Symbolic Logic* 48(4), pp. 931–940, doi:10.2307/2273659.

[2] *W3C WS-CDL*. http://www.w3.org/2002/ws/chor/.

[3] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini & Nobuko Yoshida (2014): *On the Preciseness of Subtyping in Session Types*. In: *PPDP*, ACM Press, pp. 135–146, doi:10.1145/2643135.2643138.

[4] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida & Luca Padovani (2015): *Global Progress for Dynamically Interleaved Multiparty Sessions*. *Mathematical Structures in Computer Science*, doi:10.1017/S0960129514000188. To appear.

[5] Romain Demangeon & Kohei Honda (2011): *Full Abstraction in a Subtyped pi-Calculus with Linear Types*. In: *CONCUR*, *LNCS* 6901, Springer, pp. 280–296, doi:10.1007/978-3-642-23217-6_19.

[6] Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro & Adolfo Piperno (1998): *A Filter Model for Concurrent lambda-Calculus*. *SIAM Journal on Computing* 27(5), pp. 1376–1419, doi:10.1137/S0097539794275860.

[7] Mariangiola Dezani-Ciancaglini & Silvia Ghilezan (2014): *Preciseness of Subtyping on Intersection and Union Types*. In: *RTATLCA*, *LNCS* 8560, Springer, pp. 194–207, doi:10.1007/978-3-319-08918-8_14.

[8] Simon 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.

[9] Robert Harper (2013): *Practical Foundations for Programming Languages*. Cambridge University Press.

[10] A. S. Henriksen, L. Nielsen, T. Hildebrandt, N. Yoshida & F. Henglein (2012): *Trustworthy Pervasive Healthcare Services via Multi-party Session Types*. In: *FHIES*, *LNCS* 7789, Springer, pp. 124–141, doi:10.1007/978-3-642-39088-3_8.

[11] J. Roger Hindley (1983): *The Completeness Theorem for Typing Lambda-Terms*. *Theoretical Computer Science* 22, pp. 1–17, doi:10.1016/0304-3975(83)90136-6.

[12] Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen & Nobuko Yoshida (2011): *Scribbling Interactions with a Formal Foundation*. In: *ICDCIT*, *LNCS* 6536, Springer, pp. 55–75, doi:10.1007/978-3-642-19056-8_4.

[13] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty Asynchronous Session Types*. In: *POPL*, ACM Press, pp. 273–284, doi:10.1145/1328438.1328472.

[14] Raymond Hu, Dimitrios Kouzapas, Olivier Pernet, Nobuko Yoshida & Kohei Honda (2010): *Type-Safe Eventful Sessions in Java*. In: *ECOOP*, *LNCS* 6183, Springer, pp. 329–353, doi:10.1007/978-3-642-14107-2_16.

[15] Dimitrios Kouzapas & Nobuko Yoshida (2013): *Globally Governed Session Semantics*. In: *CONCUR*, *LNCS* 8052, Springer, pp. 395–409, doi:10.1145/1328438.1328472.

[16] Jay Ligatti, Jeremy Blackburn & Michael Nachtigal (2014): *On Subtyping-Relation Completeness, with an Application to Iso-Recursive Types*. Technical Report, University of South Florida.

[17] Dimitris Mostrous, Nobuko Yoshida & Kohei Honda (2009): *Global Principal Typing in Partially Commutative Asynchronous Sessions*. In: *ESOP*, *LNCS* 5502, Springer, pp. 316–332, doi:10.1007/978-3-642-00590-9_23.

[18] Luca Padovani (2011): *Session Types = Intersection Types + Union Types*. In: *ITRS*, *EPTCS* 45, Open Publishing Association, pp. 71–89, doi:10.4204/EPTCS.45.6.

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

[20] *Savara JBoss Project*. http://www.jboss.org/savara.

[21] Kaku Takeuchi, Kohei Honda & Makoto Kubo (1994): *An Interaction-based Language and its Typing System*. In: *PARLE'94*, *LNCS* 817, pp. 398–413, doi:10.1007/3-540-58184-7_118.

[22] UNIFI (2002): *International Organization for Standardization ISO 20022 UNIversal Financial Industry message scheme*. http://www.iso20022.org.

[23] Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri & Raymond Hu (2010): *Parameterised Multiparty Session Types*. In: *FOSSACS*, *LNCS* 6014, Springer, pp. 128–145, doi:10.1007/978-3-642-12032-9_10.