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 Pantovic
Univerzitet u Novom Sadu, Serbia

Nobuko Yoshida
Imperial College London

1 Introduction. 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: denotational and operational. The former 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 latter 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. Operational preciseness has been first introduced in [2] for a call-by-value \( \lambda \)-calculus with sum, product and recursive types. Both operational and denotational preciseness have been studied in [5] for a \( \lambda \)-calculus with choice and parallel constructors and in [3] for binary sessions. The result of this paper is the operational and denotational preciseness of the subtyping introduced in [4, 6] for a synchronous multiparty session calculus [8]. The most technical challenge is the completeness for the operational preciseness, which requires the construction of characteristic global types. The core of this construction is a cyclic communication between all session participants but one. We prove that, if the subtyping relation is extended, then characteristic global types lead to deadlock, violating communication safety.

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 [8] eliminating both shared channels for session initiations and session channels for communications inside sessions.

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 \( n \), \( i \), and boolean values \( \text{true} \) and \( \text{false} \). The expressions \( e \) are variables or values or expressions built from expressions by applying the operators \( \text{succ}, \text{neg}, \neg, \oplus \), or the relation \( > \).

Processes \( P \) are defined by:

\[
P ::= \ p?\ell(x).P \mid p!\ell(e).P \mid P + P \mid \text{if } e \text{ then } P \text{ else } P \mid \mu X.P \mid X \mid 0
\]

Session communications are performed between an input process \( p?\ell(x).P \) and an output process \( q!\ell(e).Q \), where the participant \( p \) sends the value of expression \( e \) with label \( \ell \) to participant \( q \).
Precise subtyping for synchronous multiparty sessions

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 of types, 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. $\mu X.P$ is not a process.

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

$$\mathcal{M} ::= p \triangleleft P | \mathcal{M} | \mathcal{M}$$

We will use $\sum_{i \in I} P_i$ as short for $P_1 + \ldots + P_n$, and $\prod_{i \in I} p_i \triangleleft P_i$ as short for $p_1 \triangleleft P_1 | \ldots | p_n \triangleleft P_n$, where $I = \{1, \ldots, n\}$.

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

The computational rules of multiparty sessions include the standard rules for conditionals plus the rule for communication:

$$[\text{R-COMM}] \quad \begin{array}{c} j \in I \quad e \downarrow v \\ p \triangleleft \sum_{i \in I} q?\ell_i(x).P_i & q \triangleleft p!\ell_j(e).Q \rightarrow p \triangleleft P_j\{v/x\} & q \triangleleft Q \end{array}$$

In this rule participant $q$ sends the value $v$ choosing label $\ell_j$ to participant $p$ which offers inputs on all labels $\ell_i$ with $i \in I$. The computational rules are closed with respect to the structural congruence defined as expected and the following reduction contexts:

$$\mathcal{C}[\cdot] ::= [\cdot] | \mathcal{C}[\cdot] | \mathcal{M}$$

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 $\mathcal{M}$ is stuck if $\mathcal{M} \not\equiv p \triangleleft 0$ and there is no multiparty session $\mathcal{M}'$ such that $\mathcal{M} \rightarrow \mathcal{M}'$. A multiparty session $\mathcal{M}$ gets stuck, notation $\text{stuck}(\mathcal{M})$, if it reduces to a stuck multiparty session.

3 Type System. This section introduces the type system, which is a simplification of that in [8] due to the new formulation of the calculus.

Types Sorts are ranged over by $S$ and defined by: $S ::= \text{nat} | \text{int} | \text{bool}$

Global types generated by:

$$G ::= p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I} | \mu t.G | t | \text{end}$$

(with $\ell_i \neq \ell_j$ for $i \neq j$ and $i, j \in I$) describe the whole conversation scenarios of multiparty sessions. Session Types correspond to projections of global types on the individual participants. Formally the projection of the global type $G$ onto the participant $r$, notation $G \upharpoonright r$, is given by:
Subtyping

The grammar of session types, ranged over by \( \Gamma \) where

\[
\forall i \in I: \quad S_i \leq S_i' \quad T_i \leq T_i'
\]

\[
\bigwedge_{i \in I} p?\ell_i(S_i).T_i \leq \bigwedge_{i \in I} p?\ell_i(S_i').T_i'
\]

\[
\bigvee_{i \in I} p!\ell_i(S_i).T_i \leq \bigvee_{i \in I} p!\ell_i(S_i').T_i'
\]

Table 1: Subtyping rules.

\[
p \rightarrow q : \{ \ell_i(S_i).G_i \}_{i \in I} \mid r = \begin{cases} \bigvee_{i \in I} q!\ell_i(S_i).G_i \mid r & \text{if } p = r, \\ \bigwedge_{i \in I} p?\ell_i(S_i).G_i \mid r & \text{if } q = r, \\ \bigwedge_{i \in I} G_i \mid r & \text{if } p \neq r, q \neq r \text{ and } G_i \mid r = G_j \mid r \text{ for each } i, j \in I \\ G \mid r & \text{if } p \neq r, q \neq r \text{ and } G_i \mid r \text{ is an input type for each } i \in I. \end{cases}
\]

\[
\mu T \mid r = \begin{cases} G \mid r & \text{if } r \text{ occurs in } G, \\ \text{end} & \text{otherwise.} \end{cases}
\]

The grammar of session types, ranged over by \( T \), is then

\[
T ::= \bigwedge_{i \in I} p?\ell_i(S_i).T_i \mid \bigvee_{i \in I} q!\ell_i(S_i).T_i \mid \mu T \mid \text{end}
\]

Subtyping \( \leq : \) on sorts is the minimal reflexive and transitive closure of the relation induced by the rule: \( \text{nat} \leq : \text{int} \). Subtyping \( \leq : \) on session types takes into account the contra-variance of input, the covariance of outputs [6], and the standard rules for intersection and union. Table 1 gives the subtyping rules: the double line in rules indicates that the rules are interpreted coinductively [9 21.1]).

Typing system

We distinguish three kinds of typing judgments

\[
\Gamma \vdash e : S \quad \Gamma \vdash P : T \quad \vdash M
\]

where \( \Gamma \) is the environment \( \Gamma ::= \emptyset \mid \Gamma, x : S \mid \Gamma, X : T \) that associates expression variables with sorts and process variables with session types. The typing rules for expressions are standard. Table 2 gives the remaining typing rules. Processes are typed as expected, we only notice that the syntax of session types only allows input processes in external choices and output processes in the branches of conditionals. In order to type a session, rule \([T-\text{sess}]\) requires that the processes which want to communicate, i.e. that have a type different from end, can play as participants of a whole communication protocol, i.e. their types are projections of an unique global type. This is assured by the condition coherent\(\{(T_1,p_1),\ldots,(T_n,p_n)\}\). More precisely we define the set \( \text{pt}\{G\} \) of participants of a global type \( G \) as follows

\[
\text{pt}\{p \rightarrow q : \{ \ell_i(S_i).G_i \}_{i \in I} \} = \{ p, q \} \cup \text{pt}\{G_i\} \quad \{ i \in I \}
\]

\[
\text{pt}\{\mu T\} = \text{pt}\{G\} \quad \text{pt}\{t\} = \emptyset \quad \text{pt}\{\text{end}\} = \emptyset
\]

and we say that \( \{(T_1,p_1),\ldots,(T_n,p_n)\} \) is coherent (notation coherent\(\{(T_1,p_1),\ldots,(T_n,p_n)\}\)) if there is a global type \( G \) with \( \text{pt}\{G\} = \{ p_1, p_2, \ldots, p_n \} \) and \( T_i = G \mid p_i \), \( i = 1, \ldots, n \).

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.

**Theorem 3.1 (Subject reduction)** If \( \vdash \mathcal{M} \) and \( \mathcal{M} \rightarrow \mathcal{M}' \), then \( \vdash \mathcal{M}' \).

**Theorem 3.2 (Safety)** If \( \vdash \mathcal{M} \), then it does not hold stuck(\( \mathcal{M} \)).

\[1\] The projectability of \( G \) assures \( \text{pt}\{G\} = \text{pt}\{G_j\} \) for all \( i, j \in I \).
Precise subtyping for synchronous multiparty sessions

\[
\begin{align*}
\text{[T-IN]} & \quad \Gamma, x : S \vdash P : T \\
\text{[T-OUT]} & \quad \Gamma \vdash q\ell(x).P : q\ell(S).T \\
\text{[T-CHOICE]} & \quad \Gamma \vdash P_1 : T_1, \Gamma \vdash P_2 : T_2 \\
\text{[T-COND]} & \quad \Gamma \vdash e : \text{bool} \quad \Gamma \vdash P_1 : T_1, \Gamma \vdash P_2 : T_2 \\
\text{[T-VAR]} & \quad \Gamma \vdash X \\
\text{[T-REC]} & \quad \Gamma, X : T \vdash \mu X.P : T \\
\text{[T-0]} & \quad \Gamma \vdash 0 : \text{end} \\
\text{[T-SUB]} & \quad \Gamma \vdash P : T \quad T \leq T' \\
\end{align*}
\]

\[\vdash p_1 < P_1 \mid \ldots \mid p_n < P_n \mid q_1 < Q_1 \mid \ldots \mid q_m < Q_m\]

Table 2: Typing rules for processes and sessions.

4 Operational Preciseness. We adapt the notion of operational preciseness \[2, 3, 5\] to our calculus.

Definition 4.1 A subtyping relation is operationally precise if for any two types \(T\) and \(T'\) the following equivalence holds:
\[T \leq T' \iff (\exists P, T, \mathcal{M} \mid \Gamma \vdash P : T \land \forall Q : T' \vdash \Gamma \vdash Q : T' \Rightarrow \vdash p < Q \mid \mathcal{M} \text{ and } \text{stuck}(p < Q \mid \mathcal{M}).\]

The operational soundness, i.e., if for all \(Q\) such that \(\vdash Q : T'\) implies \(\vdash p < Q \mid \mathcal{M}\), then \(p < Q \mid \mathcal{M}\) is not stuck, follows from the subsumption rule \([\text{T-SUB}]\) and the safety theorem, Theorem 3.2.

To show the vice versa, it is handy to define the set \(pt\{T\}\) of participants of a session type \(T\) as follows
\[pt\{\bigwedge_{i \in I} p\ell_i(S_i).T_i\} = pt\{\bigvee_{i \in I} p!\ell_i(S_i).T_i\} = \{p\} \cup \bigcup_{i \in I} pt\{T_i\}\]
\[pt\{\mu T\} = pt\{T\} \quad pt\{t\} = pt\{\text{end}\} = \emptyset\]

The proof of operational completeness comes in four steps.

- **Step 1** We characterise the negation of the subtyping relation by inductive rules (notation \(\not\leq\)).
- **Step 2** For each type \(T\) and participant \(p \not\in pt\{T\}\), we define a characteristic global type \(G'(T, p)\) such that \(\not\leq\{T, p\} \vdash p = T\).
- **Step 3** For each type \(T\), we define a characteristic process \(G(T)\) typed by \(T\), which offers the series of interactions described by \(T\).
- **Step 4** We prove that if \(T \not\leq T'\), then \(\text{stuck}(p < G'(T)| \prod_{1 \leq i \leq n} p_i < G'(T_i))\), where \(G = G'(T', p)\), \(pt\{T'\} = \{p_1, \ldots, p_n\}\), and \(T_i = G \mid p_i\) for \(1 \leq i \leq n\). Hence we achieve completeness by choosing \(P = G(T)\) and \(\mathcal{M} = \prod_{1 \leq i \leq n} p_i < G'(T_i)\) in the definition of preciseness (Definition 4.1).

Negation of subtyping. Table 3 gives the negation of subtyping, which uses the negation of suborting \(\not\leq\) 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, 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 \(T \not\leq T'\) or \(T \leq T'\) holds for two arbitrary types \(T, T'\) by giving a decision algorithm.

**Characteristic global types** The characteristic global type \(G(T, p)\) of the type \(T\) for the participant \(p\) describes not only the communications between \(p\) and all participants in \(pt\{T\}\) following \(T\). In fact after each communication involving \(p\) and some \(q \in pt\{T\}\), \(q\) starts a cyclic communication involving...
all participants in \( pt\{T\} \) both as receivers and senders. This is needed for getting both a projectable global type and a stuck session for example when two outputs with different receivers are permuted, see below. More precisely, we define the characteristic global type \( G(T,p) \) of the type \( T \) for the participant \( p \not\in pt\{T\} \) as \( G(T,p) = G_0(T,p,pt\{T\}) \) where:

\[
\begin{align*}
G_0(\bigwedge_{i \in I} p_{j_0} ? \ell_i(S_i) \cdot T_i, p, \{p_j\}_{1 \leq j \leq n}) &= p_{j_0} \rightarrow p : \{\ell_i(S_i).G_i^{p_j}\}_{i \in I} \\
G_0(\bigvee_{i \in I} p_{j_0} ? \ell_i(S_i) \cdot T_i, p, \{p_j\}_{1 \leq j \leq n}) &= p \rightarrow p_{j_0} : \{\ell_i(S_i).G_i^{p_j}\}_{i \in I} \\
G_0(\mu t.T.p, \{p_j\}_{1 \leq j \leq n}) &= \mu t.G_0(T,p,\{p_j\}_{1 \leq j \leq n}) \\
G_0(\text{end}, p, \{p_j\}_{1 \leq j \leq n}) &= \text{end}
\end{align*}
\]

(omitting unnecessary brackets)

\[
G_i^{p_j} = p_{j_0} \rightarrow p_{j_{0}+1} : \ell_i(\text{bool}) \ldots p_{n-1} \rightarrow p_n : \ell_i(\text{bool}).p_n \rightarrow p_1 : \ell_i(\text{bool}).
\]

\[
p_1 \rightarrow p_2 : \ell_i(\text{bool}) \ldots p_{j_0-1} \rightarrow p_{j_0} : \ell_i(\text{bool}).G_0(T_i, p, \{p_j\}_{1 \leq j \leq n})
\]

It is easy to verify that \( G(T,p) \mid p = T \) by induction on the definition of characteristic global types.

**Characteristic processes** We define the characteristic process \( \mathcal{P}(T) \) of the type \( T \) by using the operators \( \text{succ} \), \( \text{neg} \), 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. The definition of \( \mathcal{P}(T) \) by induction on \( T \) is as follows:

\[
\mathcal{P}(T) = \begin{cases}
p ? \ell(x).\text{if } \text{succ}(x) > 0 \text{ then } \mathcal{P}(T') \text{ else } \mathcal{P}(T') & \text{if } T = p ? \ell(nat).T', \\
p ? \ell(x).\text{if } \text{neg}(x) > 0 \text{ then } \mathcal{P}(T') \text{ else } \mathcal{P}(T') & \text{if } T = p ? \ell(int).T', \\
p ? \ell(x).\text{if } \neg x \text{ then } \mathcal{P}(T') \text{ else } \mathcal{P}(T') & \text{if } T = p ? \ell(bool).T', \\
p! \ell(S).\mathcal{P}(T') & \text{if } T = \mathcal{P}(T'), \\
p! \ell(-S).\mathcal{P}(T') & \text{if } T = \mathcal{P}(T'), \\
p! \ell(\text{true}).\mathcal{P}(T') & \text{if } T = \mathcal{P}(T'), \\
p! \ell(\text{false}).\mathcal{P}(T') & \text{if } T = \mathcal{P}(T'), \\
\mathcal{P}(T_1) + \mathcal{P}(T_2) & \text{if } T = T_1 \lor T_2, \\
\mu X_1.\mathcal{P}(T') & \text{if } T = \mu X.\mathcal{P}(T'), \\
X_1 & \text{if } T = \text{end},
\end{cases}
\]
By induction on the structure of $\mathcal{P}(T)$ it is easy to verify that $\vdash \mathcal{P}(T) : T$.

We have now all necessary definitions to show preciseness of subtyping.

**Theorem 4.1 (Preciseness)** The synchronous multiparty session subtyping is precise.

An example showing the utility of the cyclic communication in the definition of characteristic global types is $T = p_1!\ell_1(\text{nat}).p_2!\ell_2(\text{nat}).\text{end}$ and $T' = p_2!\ell_2(\text{nat}).p_1!\ell_1(\text{nat}).\text{end}$.

In fact without the cyclic communication the characteristic global type of $T'$ would be $G = p \rightarrow p_2 : \ell_2(\text{nat}).p \rightarrow p_1 : \ell_1(\text{nat}).\text{end}$ and then $\mathcal{M} = p_1 \triangleleft \mathcal{P}(G \upharpoonright p_1) \triangleright p_2 \triangleleft \mathcal{P}(G \upharpoonright p_2) = p_1 \triangleleft p?\ell_1(x).0 \triangleright p_2 \triangleleft p?\ell_2(x).0$. Being $\mathcal{P}(T) = p_1!\ell_1(5).p_2!\ell_2(5).0$, the session $p \triangleleft \mathcal{P}(T)$, $\mathcal{M}$ reduces to $p \triangleleft 0$. Instead

$$\mathcal{G}(T', p) = \begin{cases} p \rightarrow p_2 : \ell_2(\text{nat}).p_1 \rightarrow p_1 : \ell_2(\text{bool}).p_2 \rightarrow p_2 : \ell_2(\text{bool}). \end{cases}$$

which implies $\mathcal{P}(\mathcal{G}(T', p) \upharpoonright p_1) = p_2?\ell_2(x)$. . . . and $\mathcal{P}(\mathcal{G}(T', p) \upharpoonright p_2) = p?\ell_2(x)$. . . . It is then easy to verify that $p \triangleleft \mathcal{P}(T) \upharpoonright p_1 \triangleleft \mathcal{P}(\mathcal{G}(T', p) \upharpoonright p_1) \triangleright p_2 \triangleleft \mathcal{P}(\mathcal{G}(T', p) \upharpoonright p_2)$ is stuck.

5 Denotational Preciseness. In $\lambda$-calculus types are usually interpreted as subsets of the domains of $\lambda$-models [1,7]. Denotational preciseness of subtyping is then:

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

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.

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

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

$$\vdash \mathcal{P}(T) : T' \text{ implies } T \subseteq T',$$

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

$$p \triangleleft \mathcal{P}(T) \upharpoonright \prod_{1 \leq i \leq n} \mathcal{P}(T_i)$$

where $\text{pt}\{T\} = \{ p_i \}_{1 \leq i \leq n}$ and $G = \mathcal{G}(T', p)$ and $T_i = G \upharpoonright p_i$ for $1 \leq i \leq n$ could be typed. Theorem 4.1 shows that this process is stuck, and this contradicts soundness of the type systems. We get the desired property, which implies denotational completeness, since if $T \not\subseteq T'$, then $\mathcal{P}(T) \in [T]$, but $\mathcal{P}(T) \not\subseteq [T']$.

**Theorem 5.1 (Denotational preciseness)** The subtyping relations is denotationally precise.

REFERENCES.


A Missing Definitions. Tables 4, 5 and 6 give the operational semantics of the multiparty session calculus. Table 7 lists the typing rules for expressions.

<table>
<thead>
<tr>
<th>succ(n) ↓ (n + 1)</th>
<th>neg(i) ↓ (−i)</th>
<th>−true ↓ false</th>
<th>−false ↓ true</th>
<th>v ↓ v</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i_1 &gt; i_2) ↓</td>
<td>true if i_1 &gt; i_2,</td>
<td>e_1 ↓ v or e_2 ↓ v</td>
<td>e ↓ v</td>
<td>E(v) ↓ v'</td>
</tr>
<tr>
<td></td>
<td>false otherwise</td>
<td>e_1 ⊕ e_2 ↓ v</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Expression evaluation.

<table>
<thead>
<tr>
<th>[S-EXCH 1]</th>
<th>[S-EXCH 2]</th>
<th>[S-MULTI]</th>
</tr>
</thead>
<tbody>
<tr>
<td>P + Q ≡ Q + P</td>
<td>(P + Q) + R ≡ P + (Q + R)</td>
<td>P ≡ Q ⇒ p &lt; P ≡ p &lt; Q</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[S-PAR 1]</th>
<th>[S-PAR 2]</th>
<th>[S-PAR 3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>p &lt; 0</td>
<td>M ≡ M</td>
<td>M</td>
</tr>
</tbody>
</table>

Table 5: Structural congruence.

<table>
<thead>
<tr>
<th>[R-COMM]</th>
<th>j ∈ I</th>
<th>e ↓ v</th>
<th>[T-CONDITIONAL]</th>
<th>e ↓ true</th>
</tr>
</thead>
<tbody>
<tr>
<td>p &lt; \sum_{i ∈ I} q_i \ell_i(x).P_i</td>
<td>q &lt; p \ell_j(e).Q</td>
<td>p &lt; \sum_{i ∈ I} q_i \ell_i(x).P_i</td>
<td>p &lt; P {v/x}</td>
<td>q &lt; Q</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[F-CONDITIONAL]</th>
<th>[R-CONTEXT]</th>
<th>[R-STRUCT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>e ↓ false</td>
<td>M → M'</td>
<td>M_1 ≡ M_1</td>
</tr>
<tr>
<td>p &lt; if e then P else Q</td>
<td>M → M'</td>
<td>M_1 → M_2</td>
</tr>
</tbody>
</table>

Table 6: Reduction rules.

B Proofs of Section 3

Lemma B.1 (Inversion lemma)

1. Let \( \Gamma \vdash P : T \).
   (a) If \( P = q ? \ell(x).Q \), then \( q ? \ell(S').T' \leq T \) and \( \Gamma, x : S' \vdash Q : T' \).
   (b) If \( P = q ! \ell(e).Q \), then \( q ! \ell(S').T' \leq T \), \( \Gamma \vdash e : S' \) and \( \Gamma \vdash Q : T' \).
   (c) If \( P = P_1 + P_2 \), then \( T_1 \land T_2 \leq T \), \( \Gamma \vdash P_1 : T_1 \) and \( \Gamma \vdash P_2 : T_2 \).
Proof. By induction on type derivations.

**Proposition B.1** If coherent\(\{(\land_{i \in I} q \ell_i(S_i), T_i, p_i), (T, q), (T_1, p_1), \ldots, (T_n, p_n)\}\) and \(p! \ell_j(S)\).\(T'\) \(\leq\) \(T\), then \(j \in I\) and \(T = \bigvee_{i \in I} p! \ell_i(S_i)\).\(T_i'\) and coherent\(\{(T_i, p), (T_i', q), (T_1, p_1), \ldots, (T_n, p_n)\}\) for all \(i \in I\).

Proof. By definition of coherence.

**Lemma B.2** (Substitution lemma) If \(\Gamma, x : S \vdash P : T\) and \(\Gamma \vdash \nu : S\), then \(\Gamma \vdash P\{\nu/x\} : T\).

Proof. By structural induction on \(P\).

**Theorem 3.1** (Subject reduction) If \(\vdash \mathcal{M}\) and \(\mathcal{M} \rightarrow \mathcal{M}'\), then \(\vdash \mathcal{M}'\).

Proof. By induction on the multiparty session reduction. We only consider the case of rule [\textit{r-comm}]. In this case

\[ \mathcal{M} \equiv p \land \sum_{i \in I} q \ell_i(x).P_i \mid q \land p! \ell_j(e).P \mid \prod_{i \in L} p_i \land Q_i \]

and

\[ \mathcal{M}' \equiv p \land \sum_{i \in I} q \ell_i(x).P_i \mid q \land P' \mid \prod_{i \in L} p_i \land Q_i, \]

where \(j \in I\), \(e \downarrow \nu\) and we assume \(Q_i \neq \emptyset\) for all \(i \in L\).

By Lemma B.1 we get \(\emptyset \vdash \sum_{i \in I} q \ell_i(x).P_i : T_i\), and \(\emptyset \vdash Q_i : T_i\) for \(i \in L\) with

coherent\(\{(T_i, p), (T_i', q_i), (T_1, p_1) \mid i \in L\}\).

By Lemma B.1 (1c) and (1a) \(\bigwedge_{i \in I} q \ell_i(S_i).T_i' \leq T_i\), which implies \(T = \bigwedge_{i \in I} q \ell_i(S_i).T_i'\) with \(S_i' \leq S_i, T_i' \leq T_i'\) for \(i \in I\) and \(i' \leq I\). By Lemma B.1 (1b) \(p! \ell_j(S)\).\(T''\) \(\leq\) \(T''\). Proposition B.1 implies \(j \in I'\) and \(T' = \bigvee_{i \in I} p! \ell_i(S_i).T_i''\) and coherent\(\{(T_i', p), (T_i'', q_i), (T_1, p_1) \mid i \in L\}\). By Lemma B.1 (1c) and (1a) we get \(x : S_j \vdash P_j : T_j\), which implies \(x : S_j' \vdash P_j : T_j'\) by rule T-SUB. From \(p! \ell_j(S)\).\(T''\) \(\leq\) \(\bigvee_{i \in I} p! \ell_i(S_i)\).\(T_i''\) we get \(S_j' \leq S_j)\) and \(T'' \leq T_j''\). By Lemma B.1 (1b) we get \(\emptyset \vdash e : S_j'\) and \(\emptyset \vdash P : T''\), which implies \(\emptyset \vdash e : S_j\) and \(\emptyset \vdash P : T_j''\) by rule T-SUB. Lastly Lemma B.2 gives \(\emptyset \vdash P_j\{\nu/x\} : T_j'.\)

The safety property that a typed multiparty session will never get stuck is a consequence of subject reduction.
Theorem 3.1 (Type safety) If $\vdash \mathcal{M}$, then it does not hold $\text{stuck}({\mathcal{M}})$.

C Proofs of Section 4

Lemma C.1 $T \not\not\not T'$ is the negation of $T \leq T'$.

Proof. We show that either $T \not\not\not T'$ or $T \leq T'$ holds for two arbitrary types $T, T'$ by giving a decision algorithm.

The case in which $T, T'$ are both end types is immediate.

In order to deal with recursion we unfold types every time we reach a $\mu$-binding.

If $T = p \ell(S).T_0$ and $T' = q \ell'(S').T'_0$, then $p \neq q$, or $\ell \neq \ell'$, or $p \ell = q \ell = p'$ and $S \leq S'$, or $p \ell = q \ell = p'$ and $S \leq S$, then $T \not\not\not T'$. Otherwise $T \leq T'$ holds if $T_0 \leq T'_0$ follows from the assumption $T \leq T$.

If $T' = T_1 \wedge T_2$, then $T \leq T'$ only if $T \leq T_1$ and $T \leq T_2$ follow from the assumption $T \leq T'$.

If $T = T_1 \vee T_2$, then $T \leq T'$ only if $T \leq T'$ and $T_2 \leq T'$ follow from the assumption $T \leq T'$.

If $T = \bigwedge_{i \in I} T_i$ and $T' = \bigvee_{j \in J} T_j$, then $T \leq T'$ only if there are $i \in I$ and $j \in J$ such that $T_i \leq T_j$ follows from the assumption $T \leq T'$. This implies that at least one between $I$ and $J$ must be a singleton set.

Lemma C.2 If $G = \mathcal{G}(T, p)$ and $\text{pt}\{T\} = \{p_j\}_{1\leq j\leq n}$ and $T_i = G \upharpoonright p_i$ for $1 \leq i \leq n$, then coherent$(\{T, p\}, \{T_1, p_1\}, \ldots, \{T_n, p_n\})$.

Proof. It is enough to observe that $\text{pt}(G) = \{p\} \cup \{p_j\}_{1\leq j\leq n}$ and $T = G \upharpoonright p$.

Theorem 4.1 (Preciseness) The synchronous multiparty session subtyping is precise.

Proof. We only need to show completeness of the synchronous multiparty session subtyping.

Let $p \notin \text{pt}\{T'\} = \{p_j\}_{1\leq j\leq n}$ and $G = \mathcal{G}(T', p)$ and $T_j = G \upharpoonright p_i$ for $1 \leq i \leq n$, then $\vdash Q : T'$ implies $\vdash p \ll Q \mid \prod_{1\leq i\leq n} p_i \ll \mathcal{P}(T_i)$ by Lemma C.2. We show that if $T \not\not\not T'$, then $\text{stuck}(p \ll \mathcal{P}(T) \upharpoonright \prod_{1\leq i\leq n} p_i \ll \mathcal{P}(T_i))$.

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

\[ \text{[NSUB-DIFF-PART]} \]
\[
q \neq p_h \quad \top, \bot \in \{?, 1\} \\
\frac{q \uparrow \ell(S).T_0 \not\not\not p_h \uparrow \ell'(S').T'_0}{p \ll \mathcal{P}(T) \upharpoonright \prod_{1\leq i\leq n} p_i \ll \mathcal{P}(T_i)}
\]

By definition $\mathcal{P}(T) = q \uparrow \ell(e).P$ for suitable $e, P$.

If $q \notin \{p_j\}_{1\leq j\leq n}$, then $\text{stuck}(p \ll \mathcal{P}(T) \upharpoonright \prod_{1\leq i\leq n} p_i \ll \mathcal{P}(T_i))$, since $\mathcal{P}(T)$ will never communicate.

Otherwise let $q = p_j$ with $1 \leq j \leq n$ and $j \neq h$. By construction $\mathcal{P}(T_h) = p_h \uparrow \ell(e_h).P_h$, where $\bar{x} = \begin{cases} ? & \text{if } \bar{x} = ! \\! & \text{if } \bar{x} = ? \end{cases}$, and $\mathcal{P}(T_k) = p_{f(k)} \uparrow \ell(x).P_k$, 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 $p \ll \mathcal{P}(T) \upharpoonright \prod_{1\leq i\leq n} p_i \ll \mathcal{P}(T_i)$ cannot reduce.

\[ \text{[NSUB-IN-IN]} \]
\[
\ell_1 \neq \ell_2 \text{ or } S_2 \not\leq S_1 \text{ or } T_1 \not\not\not T_2 \\
p_h ? \ell(S_1).T_1 \not\not\not p_h ? \ell(S_2).T_2
\]

A paradigmatic case is $\ell_1 = \ell_2 = \ell$, $S_1 = \text{nat}$, $S_2 = \text{int}$, $T_1 = T_2 = \text{end}$. By definition $\mathcal{P}(T) = p_h ? \ell(x).if \succ(x) > 0 \text{ then } 0 \text{ else } 0$, and $\mathcal{P}(T_h) = p ! \ell(-5).p_{g(h)} ! \ell(\text{true}).p_{f(h)} ? \ell(x).0$, and $\mathcal{P}(T_k) = p ! \ell(-5).p_{g(k)} ! \ell(\text{true}).p_{f(k)} ? \ell(x).0$.
p_{\ell(k)}?x.p_{g(k)}!\ell(true).0, where g(h) = \begin{cases} h + 1 & \text{if } h < n \\ 0 & \text{if } h = n \end{cases} and f is as in previous case, for 1 \leq k \leq n and k \neq h. Therefore p \triangleleft \mathcal{P}(T) \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ reduces to } p \triangleleft \mathcal{P}(\mathcal{T}) \text{ if } \text{succ}(5) > 0 \text{ then } 0 \text{ else } 0, \text{ which is stuck.}

By definition \mathcal{T}_1' and \mathcal{T}_2' must be intersections of inputs with the same sender, let it be p_h. Let G_1 = \mathcal{G}(\mathcal{T}_1', p), G_2 = \mathcal{G}(\mathcal{T}_2', p), P_h^{(1)} = \mathcal{P}(G_1 \upharpoonright p_h), P_h^{(2)} = \mathcal{P}(G_2 \upharpoonright p_h). Then by construction

\[ P_h = \mathcal{P}(\mathcal{G}(\mathcal{T}_1' \land \mathcal{T}_2', p) \upharpoonright p_h) = \text{if true } \oplus \text{false then } P_h^{(1)} \text{ else } P_h^{(2)}. \]

This implies that p \triangleleft \mathcal{P}(T) \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ reduces to } p \triangleleft \mathcal{P}(T) \mid p_h \triangleleft P_h^{(1)} \mid \prod_{1 \leq \neq h \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ and } p \triangleleft \mathcal{P}(T) \mid p_h \triangleleft P_h^{(2)} \mid \prod_{1 \leq \neq h \leq n} p_i \triangleleft \mathcal{P}(T_i). \text{ By induction either } p \triangleleft \mathcal{P}(T) \mid p_h \triangleleft P_h^{(1)} \mid \prod_{1 \leq \neq h \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ or } p \triangleleft \mathcal{P}(T) \mid p_h \triangleleft P_h^{(2)} \mid \prod_{1 \leq \neq h \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ is stuck, and therefore also } p \triangleleft \mathcal{P}(T) \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ is stuck.}

By definition \mathcal{T}_1' \cup \mathcal{T}_2' must be unions of outputs with the same receiver, let it be p_h. By definition \mathcal{P}(\mathcal{T}_1' \cup \mathcal{T}_2') = \text{if true } \oplus \text{false then } \mathcal{P}(\mathcal{T}_1) \text{ else } \mathcal{P}(\mathcal{T}_2). \text{ Then } p \triangleleft \mathcal{P}(\mathcal{T}_1' \cup \mathcal{T}_2') \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ reduces to } p \triangleleft \mathcal{P}(\mathcal{T}_1') \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ and } p \triangleleft \mathcal{P}(\mathcal{T}_2') \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i). \text{ By induction either } p \triangleleft \mathcal{P}(\mathcal{T}_1') \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ or } p \triangleleft \mathcal{P}(\mathcal{T}_2') \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ is stuck, and therefore}

\[ p \triangleleft \mathcal{P}(\mathcal{T}_1' \cup \mathcal{T}_2') \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \]

is stuck too.

If \mathcal{T}_1' \cup \mathcal{T}_2' \cup \mathcal{T}_1'' \cup \mathcal{T}_2'' \cup \mathcal{T}_1''' \cup \mathcal{T}_2''' \cup \mathcal{T}_1'''' \cup \mathcal{T}_2'''' \cup \mathcal{T}_1'''\prime \cup \mathcal{T}_2'''\prime \cup \mathcal{T}_1''''\prime \cup \mathcal{T}_2''''\prime \cup \mathcal{T}_1''''' \cup \mathcal{T}_2''''' \cup \mathcal{T}_1'''\prime\prime \cup \mathcal{T}_2'''\prime\prime \cup \mathcal{T}_1''''\prime\prime \cup \mathcal{T}_2''''\prime\prime \cup \mathcal{T}_1'''''\prime \cup \mathcal{T}_2'''''\prime \cup \mathcal{T}_1'''\prime\prime\prime \cup \mathcal{T}_2'''\prime\prime\prime \cup \mathcal{T}_1''''\prime\prime\prime \cup \mathcal{T}_2''''\prime\prime\prime \cup \mathcal{T}_1'''''\prime\prime \cup \mathcal{T}_2'''''\prime\prime \cup \mathcal{T}_1'''\prime\prime\prime\prime \cup \mathcal{T}_2'''\prime\prime\prime\prime \cup \mathcal{T}_1''''\prime\prime\prime\prime \cup \mathcal{T}_2''''\prime\prime\prime\prime \cup \mathcal{T}_1'''''\prime\prime\prime \cup \mathcal{T}_2'''''\prime\prime\prime \cup \mathcal{T}_1'''\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''\prime\prime\prime\prime\prime \cup \mathcal{T}_1''''\prime\prime\prime\prime\prime \cup \mathcal{T}_2''''\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''''\prime\prime\prime\prime \cup \mathcal{T}_2'''''\prime\prime\prime\prime \cup \mathcal{T}_1'''\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''\prime\prime\prime\prime\prime \cup \mathcal{T}_1''''\prime\prime\prime\prime\prime \cup \mathcal{T}_2''''\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''''\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''''\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1''''\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2''''\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''''\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''''\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1''''\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2''''\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''''\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''''\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1''''\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2''''\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''''\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''''\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1'''\prime\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2'''\prime\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_1''''\prime\prime\prime\prime\prime\prime\prime\prime\prime \cup \mathcal{T}_2''''\prime\prime\prime\prime\prime\prime\prime\prime\prime

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 \mathcal{T}_1' must be intersections of inputs with the same sender, let it be p_h, and \mathcal{T}_1'' must be unions of outputs with the same receiver, let it be p_k. By definition \mathcal{P}(T) = \sum_{i \in L} p_h ?x.p'_i.P'_i and \mathcal{P}(T_k) = \sum_{j \in J} p_k ?x.p''_j.P''_j and \mathcal{P}(T_p) = p_{p-1} ?x.p'_j.P'_p for 1 \leq p \leq n and p \neq k. Therefore p \triangleleft \mathcal{P}(T) \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ cannot reduce.}

If L contains more than one index and J is a singleton set, then by definition \mathcal{T}_1' must be intersections of inputs. By definition \mathcal{P}(T) = \sum_{i \in L} P'_i, where P'_i = \mathcal{P}(T'_i) for l \in L. Let us assume ad absurdum that
\[ p \triangleleft \mathcal{P}(T) \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \text{ is not stuck.} \] Then there must be \( l_0 \in L \) such that \( p \triangleleft P'_{l_0} \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \) is not stuck, contradicting the hypothesis.

If \( L \) is a singleton set and \( J \) contains more than one index, \( T_j' \) must be unions of outputs with the same receiver, let it be \( p_h \). Let \( G_j = \mathcal{G}(T_j', p) \) and \( P_h^{(j)} = \mathcal{P}(G_j \mid p_h) \). Then \( P_h = \mathcal{P}(\mathcal{G}(\bigvee_{j \in J} T_j', p) \mid p_h) = \sum_{j \in J} P_h^{(j)} \). Let us assume ad absurdum that \( p \triangleleft \mathcal{P}(T) \mid \prod_{1 \leq i \leq n} p_i \triangleleft \mathcal{P}(T_i) \) is not stuck. In this case there must be \( j_0 \in J \) such that \( p \triangleleft \mathcal{P}(T) \mid p_h \triangleleft P_h^{(j_0)} \mid \prod_{1 \leq i \neq h \leq n} p_i \triangleleft \mathcal{P}(T_i) \) is not stuck, contradicting the hypothesis.