Multiparty Session Types

Recap

We learnt about binary session types:

- Syntax of expressions, processes, and binary sessions.
- Operational semantics of binary sessions.
- Syntax of session types.
- Typing rules for expressions, processes, and binary sessions.
- Type safety theorems (Preservation and Progress).

From Binary to Multiparty

Recall we defined previously in the syntax:

$$
\begin{array}{llll}
\mathbf{p} & ::= & \mathbf{Alice} \mid \mathbf{Bob} & \text{Participant} \\
P, Q & ::= & \overline{\mathbf{p}}\,\langle e \rangle . P & \text{Message Send} \\
 & \mid & \mathbf{p}\,(x).P & \text{Message Receive} \\
 & \mid & \mathbf{p} \triangleright \{l_i : P_i\}_{i \in I} & \text{Branching} \\
 & \mid & \mathbf{p} \triangleleft l.P & \text{Selection} \\
 & \mid & \ldots & \ldots
\end{array}
$$

To extend our calculus to **Multiparty**, we need more participants:

$$
\mathbf{p} \quad ::= \quad \mathbf{Alice} \mid \mathbf{Bob} \mid \mathbf{Carol} \mid \cdots \quad \text{Participant}
$$

But is only extending participants enough?

Well-typed Session

In binary session types, we have the syntax for binary session:

$$\mathcal{M} ::= \mathbf{p} :: P \mid \mathbf{q} :: Q \quad \text{Binary Composition}$$

and the typing rule:

$$[\text{MTy}]\ \frac{\cdot \vdash P : S \qquad \cdot \vdash Q : \overline{S}}{\vdash \textbf{Alice} :: P \mid \textbf{Bob} :: Q}$$

We also need to extend the syntax of $\mathcal{M}$.

Duality Revisited

We previously defined *Duality*:

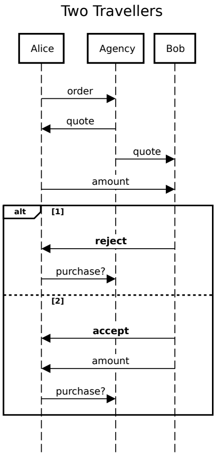
$$\textbf{Alice}\dagger = \textbf{Bob} \quad \textbf{Bob}\dagger = \textbf{Alice}$$

$$\begin{array}{rcl} \overline{\mathtt{end}} & = & \mathtt{end} \\ \overline{\mathbf{p}![U];S} & = & \mathbf{q}?[U];\overline{S} \\ \overline{\mathbf{p}?[U];S} & = & \mathbf{q}![U];\overline{S} \\ & \ldots & \end{array}$$

where $\mathbf{q} = \mathbf{p}\dagger$.

How do we define $\dagger$ beyond duality?

Travel Agency



We can have two travellers, since there can be more than two participants.

We could decompose the protocol into two binary sessions, but . . .

- Causal dependencies in messages cannot be expressed.
- n participants have up to $\mathcal{O}(n^2)$ decomposed sessions.
- Moreover . . .

Pairwise Duality Revisited

Suppose

$$
\begin{aligned}
P_{\textsf{Alice}} &= \textsf{Carol}\,(x).\overline{\textsf{Bob}}\,\langle x\rangle.\mathbf{0} &&: \textsf{Carol}?[\texttt{int}];\textsf{Bob}![\texttt{int}];\textsf{end}\\
P_{\textsf{Bob}} &= \textsf{Alice}\,(x).\overline{\textsf{Carol}}\,\langle x\rangle.\mathbf{0} &&: \textsf{Alice}?[\texttt{int}];\textsf{Carol}![\texttt{int}];\textsf{end}\\
P_{\textsf{Carol}} &= \textsf{Bob}\,(x).\overline{\textsf{Alice}}\,\langle x\rangle.\mathbf{0} &&: \textsf{Bob}?[\texttt{int}];\textsf{Alice}![\texttt{int}];\textsf{end}
\end{aligned}
$$

Pairwise, binary sessions have dual types.

Composing together, the multiparty session is stuck.

A New Methodology

When we draw the sequence diagram, we consider a global scenario.
(Global Types)

Each role can then find out their role in the global scenario.
(Local Types)

Each role can implement their own processes, independent of each other, according to their role in the global scenario.
(Local Processes)

Diagrammatically ...

$$\begin{tikzcd} & \text{Global Type } G \arrow[dl] \arrow[d, "\text{Projection}"'] \arrow[dr] & \\ \text{Local Type } S_1 \arrow[d, leftrightarrow] & \cdots \arrow[d, leftrightarrow, "\text{Typecheck}"'] & \text{Local Type } S_n \arrow[d, leftrightarrow] \\ \text{Local Process } P_1 & \cdots & \text{Local Process } P_n \end{tikzcd}$$

Syntax

As discussed previously, we extend the alphabet for participants beyond **Alice** and **Bob**, and use the same syntax for processes.

We re-define the syntax of session:

$$\begin{array}{rcll} \mathcal{M}, \mathcal{M}' & ::= & \mathbf{p} :: P & \text{Single Process} \\ & \mid & \mathcal{M} \mid \mathcal{M}' & \text{Parallel Composition} \end{array}$$

We write $\prod_{i \in I} \mathbf{p}_i :: P_i$ as the short hand notation for $\mathbf{p}_1 :: P_1 \mid \cdots \mid \mathbf{p}_n :: P_n$ for $I = \{1, \cdots, n\}$.

Structural Congruence

We adapt the usual π-calculus structural congruence rules for parallel composition into our multiparty session syntax:

$$
\begin{array}{rr}
\mathcal{M} \mid \mathcal{M}' \equiv \mathcal{M}' \mid \mathcal{M} & [\textsc{Cm-Comm}] \\
\mathcal{M}_1 \mid (\mathcal{M}_2 \mid \mathcal{M}_3) \equiv (\mathcal{M}_1 \mid \mathcal{M}_2) \mid \mathcal{M}_3 & [\textsc{Cm-Assoc}] \\
\mathsf{p} :: \mathbf{0} \mid \mathcal{M} \equiv \mathcal{M} & [\textsc{Cm-Inact}] \\
P \equiv P' \implies \mathsf{p} :: P \mid \mathcal{M} \equiv \mathsf{p} :: P' \mid \mathcal{M} & [\textsc{Cm-Ctx}]
\end{array}
$$

Operational Semantics

$$[\text{R-Com}]\ \frac{e \downarrow v \quad \mathsf{p} \neq \mathsf{q}}{\mathsf{p} :: \overline{\mathsf{q}}\,\langle e \rangle.P \mid \mathsf{q} :: \mathsf{p}\,(x).Q \mid \mathcal{M} \longrightarrow \mathsf{p} :: P \mid \mathsf{q} :: Q[v/x] \mid \mathcal{M}}$$

$$[\text{R-Label}]\ \frac{\exists j \in I. l_j = l \quad \mathsf{p} \neq \mathsf{q}}{\mathsf{p} :: \mathsf{q} \triangleleft l.P \mid \mathsf{q} :: \mathsf{p} \triangleright \{l_i : Q_i\}_{i \in I} \mid \mathcal{M} \longrightarrow \mathsf{p} :: P \mid \mathsf{q} :: Q_j \mid \mathcal{M}}$$

$$[\text{R-IfTrue}]\ \frac{e \downarrow \mathtt{true}}{\mathsf{p} :: \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \mid \mathcal{M} \longrightarrow \mathsf{p} :: P \mid \mathcal{M}}$$

$$[\text{R-IfFalse}]\ \frac{e \downarrow \mathtt{false}}{\mathsf{p} :: \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \mid \mathcal{M} \longrightarrow \mathsf{p} :: Q \mid \mathcal{M}}$$

$$[\text{R-Cong}]\ \frac{\mathcal{M}_1 \equiv \mathcal{M}_1' \quad \mathcal{M}_1' \longrightarrow \mathcal{M}_2' \quad \mathcal{M}_2' \equiv \mathcal{M}_2}{\mathcal{M}_1 \longrightarrow \mathcal{M}_2}$$

Old Travel Agency Still Works!

$\textbf{Alice} :: P_{\textsf{Alice}} \mid \textbf{Bob} :: P_{\textsf{Bob}}$ is still in valid syntax of multiparty sessions.
Binary sessions are subsumed by multiparty sessions.

What is a global type?

A global type describes the global communication behaviour between a number of participants, providing a *bird's eye view*.

Syntax

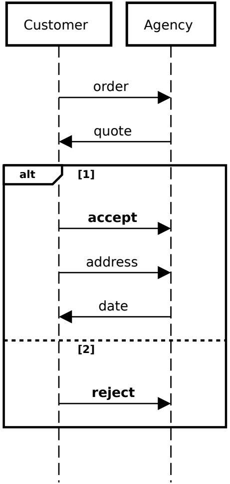
$$
\begin{array}{rcll}
G & ::= & \mathtt{end} & \text{Termination} \\
& | & \mathbf{p} \to \mathbf{q} : [U]; G & \text{Message} \\
& | & \mathbf{p} \to \mathbf{q} \, \{l_i : G_i\}_{i \in I} & \text{Branching} \\
& | & \mu \mathbf{t}.G & \text{Recursive Type} \\
& | & \mathbf{t} & \text{Type Variable}
\end{array}
$$

Travel Agency in Global Types

A global type for the travel agency protocol can be:

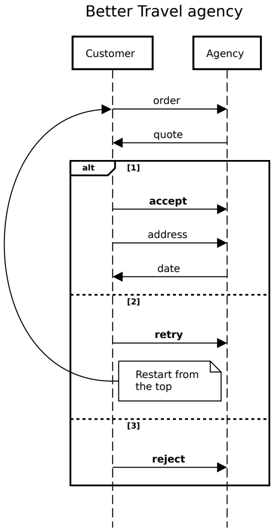
$$
\begin{array}{l}
\mathsf{Alice} \to \mathsf{Bob} : [\mathtt{string}];\\
\mathsf{Bob} \to \mathsf{Alice} : [\mathtt{int}];\\
\mathsf{Alice} \to \mathsf{Bob} \left\{ \begin{array}{l} accept : \\ \quad \mathsf{Alice} \to \mathsf{Bob} : [\mathtt{string}]; \\ \quad \mathsf{Bob} \to \mathsf{Alice} : [\mathtt{string}]; \\ \quad \mathsf{end} \\ reject : \mathsf{end} \end{array} \right\}
\end{array}
$$

Travel agency



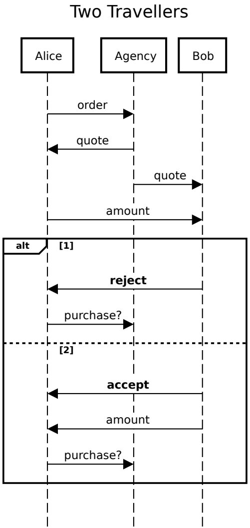
Try it Yourself: Better Travel Agency

Give the global type for the better travel agency.



Try it Yourself: Two Travellers

Give the global type for the two travellers.



Overview

Projection gives the local session types $S_{\mathbf{p}}$ for a participant $\mathbf{p}$ given a global protocol G.

We write $G \upharpoonright \mathbf{p}$ as the projection of G to the participant $\mathbf{p}$.

Intuition

To project a global type G to a participant $\mathbf{p}$, the "relevant" interaction for $\mathbf{p}$ is preserved.

$\mathbf{p} \to \mathbf{q} : [U]; \cdots$ has a prefix that $\mathbf{p}$ sends a message to $\mathbf{q}$ of sort U.

- From the viewpoint of $\mathbf{p}$, they send a message of sort U to $\mathbf{q}$, hence their local type should have a prefix $\mathbf{q}![U]; \cdots$.
- From the viewpoint of $\mathbf{q}$, they receive a message of sort U from $\mathbf{p}$, hence their local type should have a prefix $\mathbf{p}?[U]; \cdots$.
- From the viewpoint of $\mathbf{r}$, where $\mathbf{r}$ is another participant, this interaction is unrelated to them.

Intuition

$\mathbf{p} \to \mathbf{q}\ \{l_i : \cdots\}_{i \in I}$ has a prefix that **p** sends a label among a set of labels to **q**.

- From the viewpoint of **p**, they take a branch among the set of labels to **q**, hence their local type should have a prefix $\mathbf{q}{\oplus}\{l_i : \cdots\}_{i \in I}$.
- From the viewpoint of **q**, they offer branches among the set of labels from **p**, hence their local type should have a prefix $\mathbf{p}\&\{l_i : \cdots\}_{i \in I}$.
- From the viewpoint of **r**, where **r** is another participant, what is the projection for them . . .

Examples

We use a few examples to motivate the projection of branches to a participant not involved in the branch.

$$\mathsf{p} \to \mathsf{q} \left\{ \begin{array}{l} yes : \mathsf{q} \to \mathsf{r} : [\mathtt{int}]; \mathsf{end} \\ no : \mathsf{q} \to \mathsf{r} : [\mathtt{int}]; \mathsf{end} \end{array} \right\}$$

In the yes branch, **r** receives a message from **q**.
In the no branch, **r** also receives a message from **q**.

In either case, **r** receives a message from **q**, regardless of the label sent from **p** to **q**.

Examples

$$\mathbf{p} \to \mathbf{q} \begin{Bmatrix} \mathit{yes} : \mathbf{q} \to \mathbf{p} : [\mathtt{int}]; \mathbf{p} \to \mathbf{r} : [\mathtt{int}]; \mathsf{end} \\ \mathit{no} : \mathbf{q} \to \mathbf{p} : [\mathtt{string}]; \mathbf{p} \to \mathbf{r} : [\mathtt{int}]; \mathsf{end} \end{Bmatrix}$$

In the yes branch, **r** receives a message of sort `int` from **p**.
In the no branch, **r** receives a message of sort `int` from **p**.

Regardless of what branch **p** has chosen, **r** can expect a message from **p** of sort `int`.

Therefore, the global type can be projected to **r**.

Examples

$$\mathbf{p} \to \mathbf{q} \begin{Bmatrix} \mathit{yes} : \mathbf{q} \to \mathbf{r} : [\mathtt{int}]; \mathsf{end} \\ \mathit{no} : \mathsf{end} \end{Bmatrix}$$

In the yes branch, **r** receives a message from **q**.
In the no branch, **r** does nothing.

There is no way for **r** to know about the selection of **p**, which determines whether **r** needs to wait for a message from **q**.

Therefore, the global type cannot be projected to **r**.

Examples

$$\mathbf{p} \to \mathbf{q} \left\{ \begin{array}{l} yes : \mathbf{r} \to \mathbf{q} : [\mathtt{int}]; \mathsf{end} \\ no : \mathbf{r} \to \mathbf{q} : [\mathtt{string}]; \mathsf{end} \end{array} \right\}$$

In the yes branch, **r** sends a message of sort `int` to **q**.
In the no branch, **r** sends a message of sort `string` to **q**.

Whereas **q** may know the sort of the message to expect from **r**, **r** doesn't learn the choice made by **p**, and cannot always produce the correct sort according to the choice.

Therefore, the global type cannot be projected to **r**.

Plain Merge

When projecting a branch to a participant not involved, the continuations of global protocol in each branch are projected, then *merged* to a single type.

We are aware not all session types can be merged.

In *Plain Merging*, we require that the projected session types to be identical, and they merge to one session type.

Interested students can read the very gentle introduction paper to learn about *full merging* (available in materials).

Some Auxiliary Definitions

We define $\mathrm{pt}(G)$ as the set of participants involved in the global type G.

$$\begin{array}{rcl}
\mathrm{pt}(\mathbf{p} \to \mathbf{q} : [U]; G) & = & \{\mathbf{p}, \mathbf{q}\} \cup \mathrm{pt}(G) \\
\mathrm{pt}(\mathbf{p} \to \mathbf{q} \, \{l_i : G_i\}_{i \in I}) & = & \{\mathbf{p}, \mathbf{q}\} \cup \bigcup_{i \in I} \mathrm{pt}(G_i) \\
\mathrm{pt}(\mu \mathbf{t}.G) & = & \mathrm{pt}(G) \\
\mathrm{pt}(\mathbf{t}) & = & \emptyset \\
\mathrm{pt}(\mathbf{end}) & = & \emptyset
\end{array}$$

Projection, Formally

We define projection as follows:

$$\mathbf{p} \to \mathbf{q} : [U]; G \restriction \mathbf{r} \quad = \quad \begin{cases} \mathbf{q}![U]; G \restriction \mathbf{r} & \mathbf{p} = \mathbf{r} \\ \mathbf{p}?[U]; G \restriction \mathbf{r} & \mathbf{q} = \mathbf{r} \\ G \restriction \mathbf{r} & \text{otherwise} \end{cases}$$

Projection, Formally

$$\mathsf{p} \to \mathsf{q}\ \{l_i : G_i\}_{i \in I} \restriction \mathsf{r} \quad = \quad \begin{cases} \mathsf{q} \oplus \{l_i : G_i \restriction \mathsf{r}\}_{i \in I} & \mathsf{p} = \mathsf{r} \\ \mathsf{p} \& \{l_i : G_i \restriction \mathsf{r}\}_{i \in I} & \mathsf{q} = \mathsf{r} \\ G_i \restriction \mathsf{r}_{(i \in I)} & \begin{array}{l} \mathsf{p} \neq \mathsf{r}, \mathsf{q} \neq \mathsf{r} \\ \forall i, j \in I. \\ \quad G_i \restriction \mathsf{r} = G_j \restriction \mathsf{r} \end{array} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Projection, Formally

$$
\begin{aligned}
\mu \mathbf{t}.G \upharpoonright \mathsf{r} &= \begin{cases} \mathtt{end} & \mathsf{r} \notin \mathrm{pt}(G) \text{ and } \mu \mathbf{t}.G \text{ is closed} \\ \mu \mathbf{t}.G \upharpoonright \mathsf{r} & \text{otherwise} \end{cases} \\
\mathbf{t} \upharpoonright \mathsf{r} &= \mathbf{t} \\
\mathtt{end} \upharpoonright \mathsf{r} &= \mathtt{end}
\end{aligned}
$$

Exercise: Projection

We begin with a global type with only two participants.

$$G = \begin{array}{l} \textsf{Alice} \to \textsf{Bob} : [\texttt{int}]; \\ \textsf{Bob} \to \textsf{Alice} : [\texttt{bool}]; \\ \texttt{end} \end{array}$$

What is $G \upharpoonright \textsf{Alice}$ and $G \upharpoonright \textsf{Bob}$?

$$G \upharpoonright \textsf{Alice} = \textsf{Bob}![\texttt{int}]; \textsf{Bob}?[\texttt{bool}]; \texttt{end}$$
$$G \upharpoonright \textsf{Bob} = \textsf{Alice}?[\texttt{int}]; \textsf{Alice}![\texttt{bool}]; \texttt{end}$$

Note that we have $G \upharpoonright \textsf{Alice} = \overline{G \upharpoonright \textsf{Bob}}$ (using binary duality)

Exercise: Projection

$$G = \begin{array}{l} \textbf{Alice} \to \textbf{Bob} : [\texttt{int}]; \\ \textbf{Bob} \to \textbf{Carol} : [\texttt{int}]; \\ \texttt{end} \end{array}$$

What is $G \upharpoonright \textbf{Alice}$, $G \upharpoonright \textbf{Bob}$ and $G \upharpoonright \textbf{Carol}$?

$$\begin{array}{rcl} G \upharpoonright \textbf{Alice} & = & \textbf{Bob}![\texttt{int}]; \mathbf{end} \\ G \upharpoonright \textbf{Bob} & = & \textbf{Alice}?[\texttt{int}]; \textbf{Carol}![\texttt{bool}]; \mathbf{end} \\ G \upharpoonright \textbf{Carol} & = & \textbf{Bob}?[\texttt{bool}]; \mathbf{end} \end{array}$$

Verify: If you only look at communication between **Alice** and **Bob** in the projection, are they "dual" of each other?
(Similarly, for other pairs of roles)

Exercise: Projection

Are the following protocols projectable on **Carol**?

$$G_1 = \textsf{Alice} \to \textsf{Bob} \left\{ \begin{array}{l} l_1 : \textsf{Bob} \to \textsf{Carol} : [\texttt{int}]; \texttt{end} \\ l_2 : \textsf{Bob} \to \textsf{Carol} : [\texttt{string}]; \texttt{end} \end{array} \right\}$$

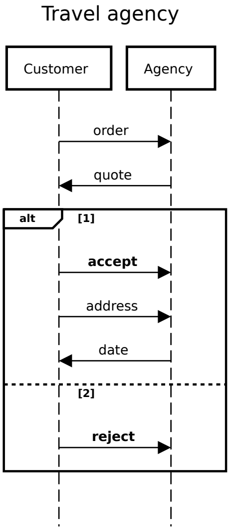
$$G_2 = \textsf{Alice} \to \textsf{Bob} \left\{ \begin{array}{l} l_1 : \textsf{Bob} \to \textsf{Carol} : [\texttt{int}]; \texttt{end} \\ l_2 : \textsf{Bob} \to \textsf{Alice} : [\texttt{int}]; \texttt{end} \end{array} \right\}$$

$$G_3 = \textsf{Alice} \to \textsf{Bob} \left\{ \begin{array}{l} l_1 : \textsf{Bob} \to \textsf{Carol} \{l_1 : \texttt{end}\} \\ l_2 : \textsf{Bob} \to \textsf{Carol} \{l_2 : \texttt{end}\} \end{array} \right\}$$

Travel Agency

$$G = \begin{array}{l} \textsf{Alice} \to \textsf{Bob} : [\texttt{string}]; \\ \textsf{Bob} \to \textsf{Alice} : [\texttt{int}]; \\ \textsf{Alice} \to \textsf{Bob} \left\{ \begin{array}{l} accept : \\ \quad \textsf{Alice} \to \textsf{Bob} : [\texttt{string}]; \\ \quad \textsf{Bob} \to \textsf{Alice} : [\texttt{string}]; \\ \quad \textsf{end} \\ reject : \textsf{end} \end{array} \right\} \end{array}$$

$$G \upharpoonright \textsf{Alice} = \begin{array}{l} \textsf{Bob}![\texttt{string}]; \textsf{Bob}?[\texttt{int}]; \\ \textsf{Bob} \oplus \left\{ \begin{array}{l} accept : \textsf{Bob}![\texttt{string}]; \\ \qquad\qquad \textsf{Bob}?[\texttt{string}]; \textsf{end} \\ reject : \textsf{end} \end{array} \right\} \end{array}$$

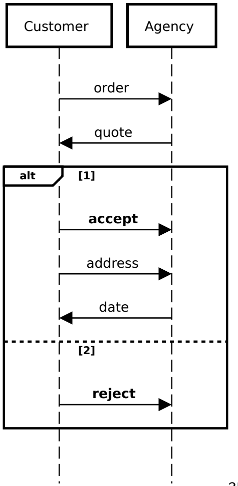


Travel Agency

$$G = \begin{array}{l} \mathsf{Alice} \to \mathsf{Bob} : [\texttt{string}]; \\ \mathsf{Bob} \to \mathsf{Alice} : [\texttt{int}]; \\ \mathsf{Alice} \to \mathsf{Bob} \left\{ \begin{array}{l} accept : \\ \quad \mathsf{Alice} \to \mathsf{Bob} : [\texttt{string}]; \\ \quad \mathsf{Bob} \to \mathsf{Alice} : [\texttt{string}]; \\ \quad \mathsf{end} \\ reject : \mathsf{end} \end{array} \right\} \end{array}$$

$$G \upharpoonright \mathsf{Bob} = \begin{array}{l} \mathsf{Alice}?[\texttt{string}]; \mathsf{Alice}![\texttt{int}]; \\ \mathsf{Alice}\& \left\{ \begin{array}{l} accept : \mathsf{Alice}?[\texttt{string}]; \\ \qquad\qquad \mathsf{Alice}![\texttt{string}]; \mathsf{end} \\ reject : \mathsf{end} \end{array} \right\} \end{array}$$

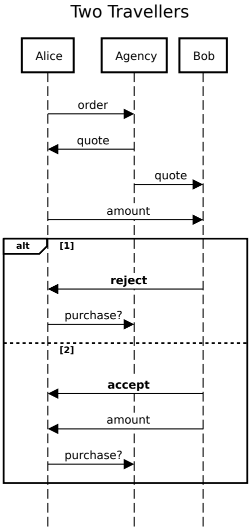
Travel agency



Try it Yourself: Two Travellers

Can you project this global type to 3 participants?

$$G = \begin{array}{l} \mathsf{Alice} \to \mathsf{Carol} : [\mathtt{string}]; \\ \mathsf{Carol} \to \mathsf{Alice} : [\mathtt{int}]; \\ \mathsf{Carol} \to \mathsf{Bob} : [\mathtt{int}]; \\ \mathsf{Alice} \to \mathsf{Bob} : [\mathtt{int}]; \\ \mathsf{Bob} \to \mathsf{Alice} \left\{ \begin{array}{l} accept : \\ \quad \mathsf{Bob} \to \mathsf{Alice} : [\mathtt{int}]; \\ \quad \mathsf{Alice} \to \mathsf{Carol} : [\mathtt{bool}]; \\ \quad \mathsf{end} \\ reject : \\ \quad \mathsf{Alice} \to \mathsf{Carol} : [\mathtt{bool}]; \\ \quad \mathsf{end} \end{array} \right\} \end{array}$$



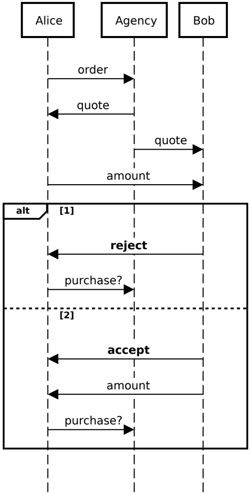
Try it Yourself: Two Travellers – Projections

$$G \upharpoonright \mathsf{Alice} = \mathsf{Carol}![\mathtt{string}]; \mathsf{Carol}?[\mathtt{int}]; \mathsf{Bob}![\mathtt{int}]; \mathsf{Bob}\& \left\{ \begin{array}{l} accept: \\ \quad \mathsf{Bob}?[\mathtt{int}]; \\ \quad \mathsf{Carol}![\mathtt{bool}]; \mathsf{end} \\ reject: \\ \quad \mathsf{Carol}![\mathtt{bool}]; \mathsf{end} \end{array} \right\}$$

$$G \upharpoonright \mathsf{Bob} = \mathsf{Carol}?[\mathtt{int}]; \mathsf{Alice}?[\mathtt{int}]; \mathsf{Alice}\oplus \left\{ \begin{array}{l} accept: \\ \quad \mathsf{Alice}![\mathtt{int}]; \mathsf{end} \\ reject: \mathsf{end} \end{array} \right\}$$

$$G \upharpoonright \mathsf{Carol} = \mathsf{Alice}?[\mathtt{string}]; \mathsf{Alice}![\mathtt{int}]; \mathsf{Bob}![\mathtt{int}]; \mathsf{Alice}?[\mathtt{bool}]; \mathsf{end}$$

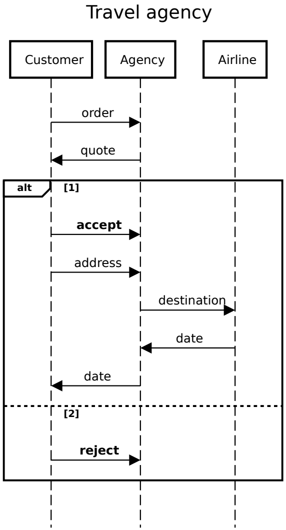
Two Travellers



Try it Yourself: Travel Agency with Airlines

Write a global type for the diagram, and project the global type to airlines.

$$
\begin{array}{l}
\mathsf{Alice} \to \mathsf{Bob} : [\mathtt{string}];\\
\mathsf{Bob} \to \mathsf{Alice} : [\mathtt{int}];\\
\mathsf{Alice} \to \mathsf{Bob} \left\{ \begin{array}{l}
\mathit{accept} : \\
\quad \mathsf{Alice} \to \mathsf{Bob} : [\mathtt{string}];\\
\quad \mathsf{Bob} \to \mathsf{Carol} : [\mathtt{string}];\\
\quad \mathsf{Carol} \to \mathsf{Bob} : [\mathtt{string}];\\
\quad \mathsf{Bob} \to \mathsf{Alice} : [\mathtt{string}];\\
\quad \mathsf{end}\\
\mathit{reject} : \mathsf{end}
\end{array} \right\}
\end{array}
$$



We cannot project the global type to airlines, because we cannot merge the two branches.

Typechecking Binary Session

Recall in binary session types, we have the judgment

$$\vdash \mathcal{M}$$

for well-typed binary session.

It can be derived via the typing rule:

$$[\text{MTy}]\ \frac{\cdot \vdash P : S \qquad \cdot \vdash Q : \overline{S}}{\vdash \textbf{Alice} :: P \mid \textbf{Bob} :: Q}$$

Typechecking Multiparty Session

For multiparty sessions, we use the global type in the judgement:

$$\vdash \mathcal{M} : G$$

It can be derived via the typing rule:

$$[\text{MTy}]\ \frac{\forall i \in I. \quad \cdot \vdash P_i : G \upharpoonright \mathsf{p}_i \qquad \operatorname{pt}(G) \subseteq \{\mathsf{p}_i \mid i \in I\}}{\vdash \prod_{i \in I} \mathsf{p}_i :: P_i : G}$$

Summary

To summarise, we discussed:

- Syntax and operational semantics for multiparty sessions
- Syntax of global types
- Projection of global types into local session types
- Typechecking multiparty sessions