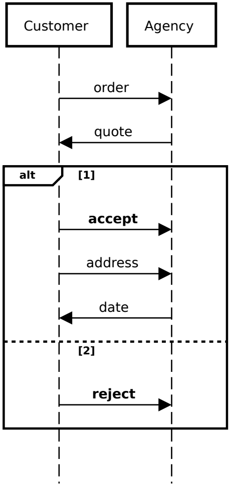
Binary Session Types

Motivating Example: Travel Agency Example

This interaction can be described as:

- The Customer posts an order
- The Agency sends a quote
- Then the customer can choose:
  - To accept the offer.
  - To simply `reject` the the quote.

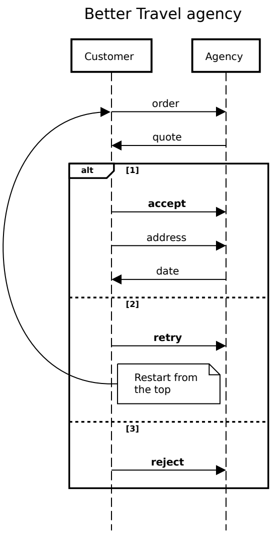
Travel agency



The Better Travel Agency Example

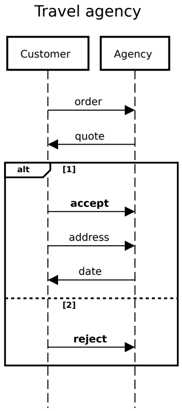
This interaction can be described as:

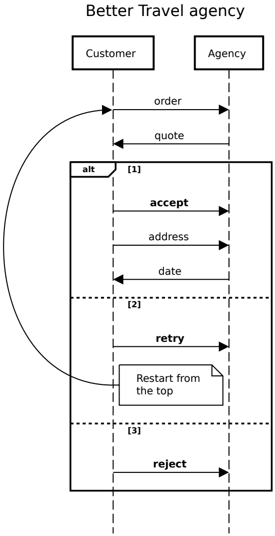
- The Customer posts an order
- The Agency sends a quote
- Then the customer can choose:
  - To `accept` the offer.
  - To `retry` with a new order (e.g.: to try a cheaper destination.)
  - To simply `reject` the the quote.



Some Sub-typing Intuition

Some of these processes seem to be related:





Some Sub-typing Intuition (2)

Dramatis Personæ:

- Alice: a customer of the old agency.
- Bob: the clerk of the old agency.
- Charlie: a customer of the better agency.
- Eve: the clerk of the better agency.

A	$\leftrightarrow$	B	✓
A	$\leftrightarrow$	E	✓
C	$\leftrightarrow$	B	✗
C	$\leftrightarrow$	E	✓

Syntax of Expressions

Before defining processes, we first introduce a simple expression language:

$$
\begin{array}{rcll}
v & ::= & \underline{n} & \text{Integers} \\
 & \mid & \texttt{true} \mid \texttt{false} & \text{Booleans} \\
 & \mid & \text{“}\texttt{str}\text{”} & \text{Strings} \\
e, e' & ::= & v & \text{Values} \\
 & \mid & x & \text{Variables} \\
 & \mid & e + e' \mid e - e' \mid -e & \text{Arithmetic} \\
 & \mid & e = e' \mid e < e' \mid e > e' & \text{Relational} \\
 & \mid & e \wedge e' \mid e \vee e' \mid \neg e & \text{Logical} \\
 & \mid & e \oplus e' & \text{Non-determinism}
\end{array}
$$

Syntax of Processes

$\mathbf{p}$	$::=$	$\mathbf{Alice} \mid \mathbf{Bob}$	Participant
P, Q	$::=$	$\mathbf{0}$	Inaction
	$\mid$	$\overline{\mathbf{p}}\,\langle e\rangle.P$	Message Send
	$\mid$	$\mathbf{p}\,(x).P$	Message Receive
	$\mid$	$\mathbf{p} \triangleright \{l_i : P_i\}_{i\in I}$	Branching
	$\mid$	$\mathbf{p} \triangleleft l.P$	Selection
	$\mid$	$\texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q$	Conditional
	$\mid$	$\mu X.P$	Recursive Process
	$\mid$	X	Process Variable
$\mathcal{M}$	$::=$	$\mathbf{p} :: P \mid \mathbf{q} :: Q$	Binary Composition

Starting Examples

$$\textsf{Alice} :: \overline{\textsf{Bob}}\,\langle 42\rangle.\mathbf{0} \mid \textsf{Bob} :: \textsf{Alice}\,(x).\mathbf{0}$$ ✓

$$\textsf{Alice} :: \textsf{Bob}\,(x).\mathbf{0} \mid \textsf{Bob} :: \overline{\textsf{Alice}}\,\langle 42\rangle.\mathbf{0}$$ ✓

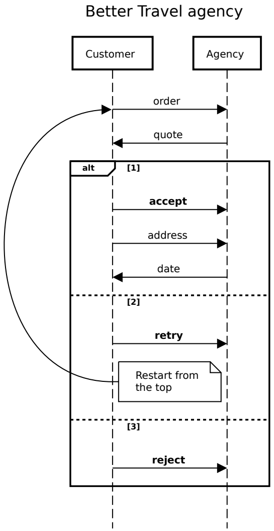
$$\textsf{Alice} :: \texttt{if true then } \overline{\textsf{Bob}}\,\langle\text{“}\texttt{hi}\text{”}\rangle.\mathbf{0} \texttt{ else } \overline{\textsf{Bob}}\,\langle\text{“}\texttt{bye}\text{”}\rangle.\mathbf{0}$$ ✓

$$\mid$$ ✓

$$\textsf{Bob} :: \textsf{Alice}\,(x).\mathbf{0}$$ ✓

$$\textsf{Alice} :: \overline{\textsf{Bob}}\,\langle 7\rangle.\mathbf{0} \mid \textsf{Bob} :: \textsf{Alice}\,(x).\overline{\textsf{Alice}}\,\langle\text{“}\texttt{thx}\text{”}\rangle.\mathbf{0}$$ ✗

Travel Agency Revisited

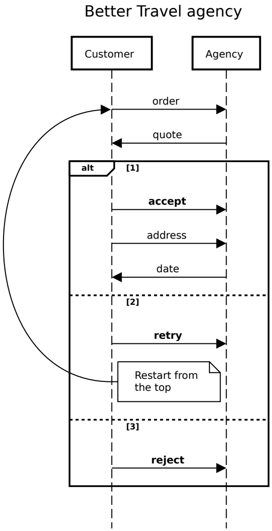


For Customer (**Alice**):

$$
\begin{array}{l}
\overline{\textbf{Bob}}\,\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\textbf{Bob}\,(quote).\\
\texttt{if}\ quote > 1000\\
\texttt{then}\ \textbf{Bob} \triangleleft retry.\\
\qquad \overline{\textbf{Bob}}\,\langle\text{“}\texttt{Florence}\text{”}\rangle.\textbf{Bob}\,(newQuote).\\
\qquad \texttt{if}\ newQuote > 1000\\
\qquad \texttt{then}\ \textbf{Bob} \triangleleft reject.\mathbf{0}\\
\qquad \texttt{else}\ \textbf{Bob} \triangleleft accept.\overline{\textbf{Bob}}\,\langle\text{“}\texttt{L'Aquila}\text{”}\rangle.\\
\qquad\qquad \textbf{Bob}\,(date).\mathbf{0}\\
\texttt{else}\ \textbf{Bob} \triangleleft accept.\overline{\textbf{Bob}}\,\langle\text{“}\texttt{L'Aquila}\text{”}\rangle.\\
\qquad \textbf{Bob}\,(date).\mathbf{0}
\end{array}
$$

We denote this process $P_{\textbf{Alice}}$.

Travel Agency Revisited



For Agency (**Bob**):

$$\mu X.\mathsf{Alice}\,(order).\overline{\mathsf{Alice}}\,\langle 1000 \oplus 5000\rangle.\mathsf{Alice} \triangleright \left\{ \begin{array}{l} retry : X \\ reject : \mathbf{0} \\ accept : \mathsf{Alice}\,(address).\overline{\mathsf{Alice}}\,\langle \text{“20230815”} \rangle.\mathbf{0} \end{array} \right\}$$

We denote this process P_{Bob}.

Composing Alice and Bob

In our calculus, we use

$$\textbf{Alice} :: P_{\textbf{Alice}} \;\Big|\; \textbf{Bob} :: P_{\textbf{Bob}}$$

to compose the two processes in parallel.

Structural Preongruence

Similar to π-calculus, we have structural congruence rules for processes and binary sessions:

$$\mu X.P \Rrightarrow P[\mu X.P/X] \quad [\textsc{C-Rec}]$$

$$\mathbf{p} :: P \mid \mathbf{q} :: Q \;\equiv\; \mathbf{q} :: Q \mid \mathbf{p} :: P \quad [\textsc{Cm-Comm}]$$

$$P \equiv P' \implies \mathbf{p} :: P \mid \mathbf{q} :: Q \;\equiv\; \mathbf{p} :: P' \mid \mathbf{q} :: Q \quad [\textsc{Cm-Ctx}]$$

The structural precongruence relation is the smallest reflexive and transitive precongruence relation containing the rules above. Where $\equiv = (\Rrightarrow \cup \Rrightarrow^{\dashv})$.

[C-Rec] explained

$\mu X.P$ is analogous to the fix-combinator in λ-calculus:

$$\begin{array}{rl} & \mu X.\overline{\textsf{Alice}}\,\langle 1\rangle.X \\ \Longrightarrow & \overline{\textsf{Alice}}\,\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\,\langle 1\rangle.X \\ \Longrightarrow & \overline{\textsf{Alice}}\,\langle 1\rangle.\overline{\textsf{Alice}}\,\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\,\langle 1\rangle.X \\ \Longrightarrow & \overline{\textsf{Alice}}\,\langle 1\rangle.\overline{\textsf{Alice}}\,\langle 1\rangle.\overline{\textsf{Alice}}\,\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\,\langle 1\rangle.X \\ \Longrightarrow & \cdots \end{array}$$

Expression Evaluation

$$[\textsc{E-NonDet-L}]\ \frac{e_1 \Downarrow v}{e_1 \oplus e_2 \Downarrow v}$$

$$[\textsc{E-NonDet-R}]\ \frac{e_2 \Downarrow v}{e_1 \oplus e_2 \Downarrow v}$$

The rest of the evaluation judgements follow the standard semantics of operators.

Operational Semantics

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

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

Operational Semantics

$$[\text{R-IfTrue}]\ \frac{e \downarrow \texttt{true}}{\mathbf{p} :: \texttt{if}\ e\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \mid \mathbf{q} :: Q \longrightarrow \mathbf{p} :: P_1 \mid \mathbf{q} :: Q}$$

$$[\text{R-IfFalse}]\ \frac{e \downarrow \texttt{false}}{\mathbf{p} :: \texttt{if}\ e\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \mid \mathbf{q} :: Q \longrightarrow \mathbf{p} :: P_2 \mid \mathbf{q} :: Q}$$

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

Travel Agency Revisited

$$
\begin{array}{rl}
 & \textsf{Alice} :: P_{\textsf{Alice}} \mid \textsf{Bob} :: P_{\textsf{Bob}} \\
= & \textsf{Alice} :: \overline{\textsf{Bob}}\,\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \mu X.\textsf{Alice}\,(order).\cdots \\
\Rrightarrow & \textsf{Alice} :: \overline{\textsf{Bob}}\,\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}\,(order).\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob}\,(quote).\cdots \mid \textsf{Bob} :: \overline{\textsf{Alice}}\,\langle 1000 \oplus 5000\rangle.\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{if}\ 1000 > 1000\ \cdots \mid \textsf{Bob} :: \textsf{Alice} \triangleright \{\cdots\} \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob} \triangleleft accept.\cdots \mid \textsf{Bob} :: \textsf{Alice} \triangleright \{accept : \cdots ; \cdots\} \\
\longrightarrow & \textsf{Alice} :: \overline{\textsf{Bob}}\,\langle\text{“}\texttt{L'Aquila}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}\,(address).\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob}\,(date).\mathbf{0} \mid \textsf{Bob} :: \overline{\textsf{Alice}}\,\langle\text{“20230815”}\rangle.\mathbf{0} \\
\longrightarrow & \textsf{Alice} :: \mathbf{0} \mid \textsf{Bob} :: \mathbf{0}
\end{array}
$$

Travel Agency Revisited

Try it yourself: How does the binary session reduce if

$$1000 \oplus 5000 \downarrow 5000$$

$$P_{\textsf{Alice}} = \begin{array}{l} \overline{\textsf{Bob}}\,\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\textsf{Bob}\,(quote). \\ \texttt{if}\ quote > 1000 \\ \texttt{then}\ \textsf{Bob} \triangleleft retry.\overline{\textsf{Bob}}\,\langle\text{“}\texttt{Florence}\text{”}\rangle.\textsf{Bob}\,(newQuote). \\ \qquad \texttt{if}\ newQuote > 1000 \\ \qquad \texttt{then}\ \textsf{Bob} \triangleleft reject.\mathbf{0} \\ \qquad \texttt{else}\ \textsf{Bob} \triangleleft accept.\overline{\textsf{Bob}}\,\langle\text{“}\texttt{L'Aquila}\text{”}\rangle.\textsf{Bob}\,(date).\mathbf{0} \\ \texttt{else}\ \textsf{Bob} \triangleleft accept.\overline{\textsf{Bob}}\,\langle\text{“}\texttt{L'Aquila}\text{”}\rangle.\textsf{Bob}\,(date).\mathbf{0} \end{array}$$

$$P_{\textsf{Bob}} = \begin{array}{l} \mu X.\textsf{Alice}\,(order).\overline{\textsf{Alice}}\,\langle 1000 \oplus 5000\rangle. \\ \qquad \textsf{Alice} \triangleright \left\{ \begin{array}{l} retry : X \\ reject : \mathbf{0} \\ accept : \textsf{Alice}\,(address).\overline{\textsf{Alice}}\,\langle\text{“}20230815\text{”}\rangle.\mathbf{0} \end{array} \right\} \end{array}$$

Recap

Last week, we discussed about:

- The syntax of binary session calculus
- The operational semantics
- Some examples
- Some intuition about Subtyping (travel agency, ATM)

Typing Expressions

Recall our expressions have the following syntax:

$$\begin{array}{rcl} v & ::= & \underline{n} \mid \texttt{true} \mid \texttt{false} \mid \text{“}\texttt{str}\text{”} \\ e, e' & ::= & v \mid x \mid e + e' \mid e - e' \mid -e \\ & \mid & e = e' \mid e < e' \mid e > e' \\ & \mid & e \wedge e' \mid e \vee e' \mid \neg e \\ & \mid & e \oplus e' \end{array}$$

We assign the following *Sorts* to expressions:

$$U ::= \texttt{int} \mid \texttt{bool} \mid \texttt{string}$$

Representing Type systems

A type system is comprised of:

- A syntax for types.
- A typing judgment that relates programs and types (and the needed assumptions on the environment).
- The inference rules that define the judgment.

Inference rules

$$[\textsc{RuleName}]\ \frac{A \text{ true} \qquad B \text{ true} \qquad C \text{ true}}{D \text{ true}}$$

Where it can be read as: A true, B true, and C true are the premises needed to establish as conclusion: D true.

$$[\textsc{Axiom}]\ \frac{}{E \text{ true}}$$

Axioms are rules whose conclusions do not have premises. (N.B.: the line may be omitted).

Derivation trees

Inference rules and axioms, naturally form derivation trees that show how a proof is constructed.
From the definition of the judgement: $A \clubsuit B$

$$[\text{R-1}]\ \frac{A \qquad B}{A \clubsuit B} \qquad [\text{A-1}]\ \frac{-}{p} \qquad [\text{A-2}]\ \frac{-}{q} \qquad [\text{A-3}]\ \frac{-}{r}$$

We can build a derivation that establishes $p \clubsuit (q \clubsuit r)$ in the following way:

$$[\text{R-1}]\ \frac{[\text{A-1}]\ \frac{-}{p} \qquad [\text{R-1}]\ \frac{[\text{A-2}]\ \frac{-}{q} \qquad [\text{A-3}]\ \frac{-}{r}}{q \clubsuit r}}{p \clubsuit (q \clubsuit r)}$$

How to type?

Typing, or typechecking is relating programs to their types.

How to type?

In a context where we record typing assumptions, we assign sorts to expressions with a judgment:

$$\boxed{\Gamma \vdash e : U}$$

We read this judgment as:

Under typing context Γ, the expression e has sort U.

Typing Context Γ

Typing contexts Γ stores information about variables and their sorts.
For the purpose for assigning sorts to expressions, we define

$$\Gamma ::= \cdot \mid \Gamma, x : U$$

$\cdot$ is an empty context
$\Gamma, x : U$ is a context Γ extended with an entry that x is of sort U
For convenience, we treat all variables as distinct and ordering in the typing context as not significant.

Typing rules define the judgment

$$[\textsc{Ty-Int}]\ \frac{}{\Gamma \vdash \underline{n} : \texttt{int}} \qquad [\textsc{Ty-Plus}]\ \frac{\Gamma \vdash e : \texttt{int} \qquad \Gamma \vdash e' : \texttt{int}}{\Gamma \vdash e + e' : \texttt{int}}$$

$$[\textsc{Ty-Less}]\ \frac{\Gamma \vdash e : \texttt{int} \qquad \Gamma \vdash e' : \texttt{int}}{\Gamma \vdash e < e' : \texttt{bool}}$$

$$[\textsc{Ty-Not}]\ \frac{\Gamma \vdash e : \texttt{bool}}{\Gamma \vdash \neg e : \texttt{bool}} \qquad [\textsc{Ty-NonDet}]\ \frac{\Gamma \vdash e : U \qquad \Gamma \vdash e' : U}{\Gamma \vdash e \oplus e' : U}$$

$$[\textsc{Ty-Var}]\ \frac{}{\Gamma, x : U \vdash x : U}$$

Examples of Expression Typing

Example of sum

$$[\textsc{Ty-Plus}]\ \frac{\cdot \vdash 3 : \texttt{int} \qquad \cdot \vdash 5 : \texttt{int}}{\cdot \vdash 3 + 5 : \texttt{int}}$$

Example of sum with a variable

$$[\textsc{Ty-Plus}]\ \frac{[\textsc{Ty-Var}]\ \dfrac{}{x{:}\texttt{int} \vdash x : \texttt{int}} \qquad x{:}\texttt{int} \vdash 5 : \texttt{int}}{x{:}\texttt{int} \vdash x + 5 : \texttt{int}}$$

Syntax of Session Types

S	$::=$	$\mathtt{end}$	Termination
	$\mid$	$\mathbf{p}![U]; S$	Value Send
	$\mid$	$\mathbf{p}?[U]; S$	Value Receive
	$\mid$	$\mathbf{p}\oplus\{l_i : S_i\}_{i\in I}$	Selection
	$\mid$	$\mathbf{p}\&\{l_i : S_i\}_{i\in I}$	Branching
	$\mid$	$\mathbf{t}$	Type Variable
	$\mid$	$\mu\mathbf{t}.S$	Recursive Type

We often omit `end`.

Examples of Session Types

1. $\textbf{Alice}![\texttt{int}];\textbf{Alice}![\texttt{string}];\textbf{Alice}?[\texttt{int}];\textbf{end}$
2. $\textbf{Alice}![\texttt{int}];\textbf{Alice}![\texttt{string}];\textbf{Alice}?[\texttt{int}];$
3. $\textbf{Bob}\&\begin{Bmatrix} orange : \textbf{Bob}![\texttt{string}];\textbf{Bob}![\texttt{int}];\textbf{end} \\ cherry : \textbf{Bob}?[\texttt{string}];\textbf{end} \\ reject : \textbf{end} \end{Bmatrix}$
4. $\textbf{Alice}![\texttt{int}];\mu\mathbf{t}.\textbf{Alice}![\texttt{string}];\textbf{Alice}?[\texttt{int}];\mathbf{t}$
5. Is this a correct type?
   $\textbf{Bob}![\texttt{int}];\textbf{Alice}![\texttt{string}];\textbf{Alice}?[\texttt{int}];\textbf{end}$
6. Is this a correct type?
   $\textbf{Alice}\oplus\begin{Bmatrix} orange : \textbf{Alice}![\texttt{string}];\textbf{Alice}![\texttt{int}];\mathbf{t} \\ cherry : \textbf{Alice}?[\texttt{string}];\textbf{end} \\ repeat : \mathbf{t} \end{Bmatrix}$

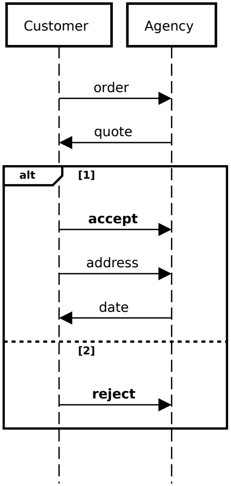
Travel Agency Revisited

Let's try to describe the travel agency with session types!

A possible type for the customer (**Alice**) is:

$$S_{\textbf{Alice}} = \begin{array}{l} \textbf{Bob}![\texttt{string}]; \\ \textbf{Bob}?[\texttt{int}]; \\ \textbf{Bob} \oplus \left\{ \begin{array}{l} \mathit{accept} : \textbf{Bob}![\texttt{string}]; \\ \qquad\qquad \textbf{Bob}?[\texttt{string}]; \mathsf{end} \\ \mathit{reject} : \mathsf{end} \end{array} \right\} \end{array}$$

Travel agency

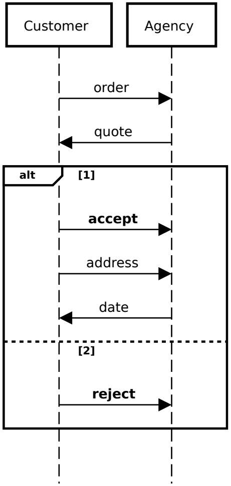


Travel Agency Revisited

A possible type for the agency (**Bob**) is:

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

Travel agency



On Recursive Types

We use an equi-recursive presentation of recursive types.

We identify $\mu\mathbf{t}.S$ and $S[\mu\mathbf{t}.S/\mathbf{t}]$
i.e. we do not distinguish between these two types.

We assume types are *closed* and *guarded*.
i.e. type variables are bounded and $\mu\mathbf{t}.\mathbf{t}$ is forbidden.

On Recursive Types and Coinduction

We can construct an infinite type with recursion.

$$\mu\mathbf{t}.\textbf{Alice}![\texttt{int}];\mathbf{t}$$

is the same type as $\textbf{Alice}![\texttt{int}];\textbf{Alice}![\texttt{int}];\textbf{Alice}![\texttt{int}];\cdots$

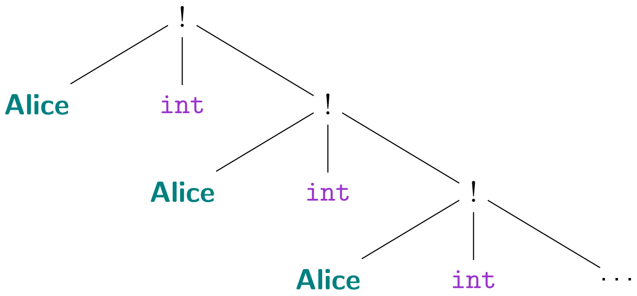
To reason about infinite types, we need to use *coinduction*.

Interested students can read [Pierce, 2002, Chapter 21] for the meta-theory of recursive types and [Kozen and Silva, 2017] for coinduction.

Recursive Types as Infinite Trees

The recursive type can also be represented as an infinite tree.

$$\mu\mathbf{t}.\mathsf{Alice}![\mathtt{int}];\mathbf{t}$$



The tree nodes will not have $\mu\mathbf{t}.S$ or $\mathbf{t}$.

Recursive Types as Infinite Trees

The following recursive types have the same tree representation (shown in previous slide).

$$\mu\mathbf{t}.\mathsf{Alice}![\mathtt{int}];\mathbf{t}$$

$$\mathsf{Alice}![\mathtt{int}];\mu\mathbf{t}.\mathsf{Alice}![\mathtt{int}];\mathbf{t}$$

$$\mu\mathbf{t}.\mathsf{Alice}![\mathtt{int}];\mathsf{Alice}![\mathtt{int}];\mathbf{t}$$

We treat them as the same type.

Labelled Transition System (LTS) of Session Types

Session types are *behavioural types* so that we can define the transition systems. We first define the actions:

$$\alpha ::= \mathbf{p}?[U] \mid \mathbf{p}![U] \mid \mathbf{p} \oplus l \mid \mathbf{p}\& l$$

The labelled transition relation on types is defined by:

$$[\text{L-In}]\ \mathbf{p}?[U];S \xrightarrow{\mathbf{p}?[U]} S \qquad [\text{L-Out}]\ \mathbf{p}![U];S \xrightarrow{\mathbf{p}![U]} S$$

$$[\text{L-Sel}]\ \frac{k \in I}{\mathbf{p}\oplus\{l_i : S_i\}_{i\in I} \xrightarrow{\mathbf{p}\oplus l_k} S_k} \qquad [\text{L-Bra}]\ \frac{k \in I}{\mathbf{p}\&\{l_i : S_i\}_{i\in I} \xrightarrow{\mathbf{p}\& l_k} S_k}$$

$$[\text{L-Rec}]\ \frac{S[\mu\mathbf{t}.S/\mathbf{t}] \xrightarrow{\alpha} S'}{\mu\mathbf{t}.S \xrightarrow{\alpha} S'}$$

Bisimulation on Session Types

Let $\mathcal{S}$ be a set of closed session types. A binary relation $\mathcal{R} \subseteq (\mathcal{S} \times \mathcal{S})$ is called *bisimulation* whenever $S_1 \mathrel{\mathcal{R}} S_2$ implies that:

- for all α, if $S_1 \xrightarrow{\alpha} S_1'$, then there exists S_2' such that $S_2 \xrightarrow{\alpha} S_2'$ and $S_1' \mathrel{\mathcal{R}} S_2'$; and
- for all α, if $S_2 \xrightarrow{\alpha} S_2'$, then there exists S_1' such that $S_1 \xrightarrow{\alpha} S_1'$ and $S_1' \mathrel{\mathcal{R}} S_2'$.

The largest bisimulation, denoted by $\sim$, is called *bisimilarity*. In this course, if $S_1 \sim S_2$, we say S_1 and S_2 are equivalent.

Examples of Recursive Types

1. $\mu\mathbf{t}.\mathbf{t}$
2. $\textsf{Alice}![\texttt{int}];\textsf{Alice}?[\texttt{bool}];\mu\mathbf{t}.\textsf{Alice}![\texttt{int}];\textsf{Alice}?[\texttt{bool}];\mathbf{t}$
3. $\mu\mathbf{t}.\textsf{Alice}![\texttt{int}];\textsf{Alice}?[\texttt{bool}];\textsf{Alice}![\texttt{int}];\textsf{Alice}?[\texttt{bool}];\mathbf{t}$
4. $\textsf{Alice}![\texttt{int}];\mu\mathbf{t}.\textsf{Alice}?[\texttt{bool}];\textsf{Alice}![\texttt{int}];\mathbf{t}$
5. $\textsf{Alice}![\texttt{int}];\mu\mathbf{t}.\textsf{Alice}?[\texttt{bool}];\textsf{Alice}![\texttt{int}];\texttt{end}$
6. $\mu\mathbf{t}.\texttt{end}$
7. $\mu\mathbf{t}.\textsf{Alice}?[\mathbf{t}];$

Challenge: Type Equivalence

Let:

$$S_1 = \textbf{Alice}![\texttt{int}];\textbf{Alice}?[\texttt{bool}];\mu\mathbf{t}.\textbf{Alice}![\texttt{int}];\textbf{Alice}?[\texttt{bool}];\mathbf{t}$$

$$S_2 = \mu\mathbf{t}.\textbf{Alice}![\texttt{int}];\textbf{Alice}?[\texttt{bool}];\textbf{Alice}![\texttt{int}];\textbf{Alice}?[\texttt{bool}];\mathbf{t}$$

$$S_3 = \textbf{Alice}![\texttt{int}];\mu\mathbf{t}.\textbf{Alice}?[\texttt{bool}];\textbf{Alice}![\texttt{int}];\mathbf{t}$$

$$S_4 = \mu\mathbf{t}.\textbf{Alice}![\texttt{int}];\textbf{Alice}?[\texttt{bool}];\textbf{Alice}![\texttt{int}];\mathbf{t}$$

Can you prove $S_1 \sim S_2$ and $S_2 \sim S_3$? Can you prove $S_3 \not\sim S_4$?

Motivation

We use session types to describe the behaviour of processes.

To achieve type safety, we would like to ensure that the binary session is able to progress.

Recall, in the operational semantics, if a process sends and the other receives, the binary session reduces. (Similarly for offering and taking branches)

We define *duality* to describe such relation of session types. Duality becomes important when we type a binary session.

Duality of Binary Session Types

We first define *duality* of participants:

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

Duality of Binary Session Types

We define duality of binary session types as a *function*:

$$\begin{array}{rcl}
\overline{\mathtt{end}} & = & \mathtt{end} \\
\overline{\mathbf{p}![U];S} & = & \mathbf{q}?[U];\overline{S} \\
\overline{\mathbf{p}?[U];S} & = & \mathbf{q}![U];\overline{S} \\
\overline{\mathbf{p}\oplus\{l_i : S_i\}_{i\in I}} & = & \mathbf{q}\&\{l_i : \overline{S_i}\}_{i\in I} \\
\overline{\mathbf{p}\&\{l_i : S_i\}_{i\in I}} & = & \mathbf{q}\oplus\{l_i : \overline{S_i}\}_{i\in I} \\
\overline{\mathbf{t}} & = & \mathbf{t} \\
\overline{\mu\mathbf{t}.S} & = & \mu\mathbf{t}.\overline{S}
\end{array}$$

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

Exercise: Show that $\overline{\overline{S}} = S$.

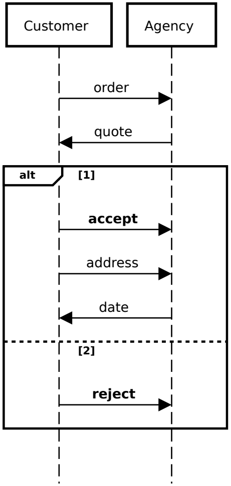
Travel Agency Revisited

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

$$S_{\textsf{Bob}} = \begin{array}{l} \textsf{Alice}?[\texttt{string}]; \\ \textsf{Alice}![\texttt{int}]; \\ \textsf{Alice}\&\left\{\begin{array}{l} \mathit{accept} : \textsf{Alice}?[\texttt{string}]; \\ \qquad\qquad \textsf{Alice}![\texttt{string}]; \mathtt{end} \\ \mathit{reject} : \mathtt{end} \end{array}\right\} \end{array}$$

Verify: $S_{\textsf{Alice}} = \overline{S_{\textsf{Bob}}}$.

Travel agency



Duality operates on the infinite tree

We are usually interested in determining whether a type is a dual of the other type.

For example: Let

$$
\begin{aligned}
S_1 &= \textbf{Alice}![\texttt{int}]; \mu \mathbf{t}.\textbf{Alice}?[\texttt{bool}]; \textbf{Alice}![\texttt{int}]; \mathbf{t} \\
S_2 &= \mu \mathbf{t}.\textbf{Bob}?[\texttt{int}]; \textbf{Bob}![\texttt{bool}]; \mathbf{t}
\end{aligned}
$$

Is S_1 dual of S_2?

Naively

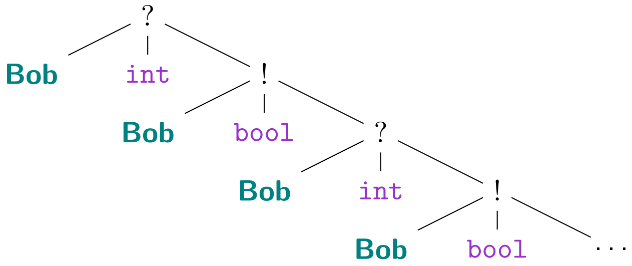
$$\overline{S_1} = \textbf{Bob}?[\texttt{int}]; \mu \mathbf{t}.\textbf{Bob}![\texttt{bool}]; \textbf{Bob}?[\texttt{int}]; \mathbf{t}$$

Duality operates on the infinite tree

We have that

$$\begin{aligned}
\overline{S_1} &= \textbf{Bob}?[\texttt{int}];\mu\mathbf{t}.\textbf{Bob}![\texttt{bool}];\textbf{Bob}?[\texttt{int}];\mathbf{t} \\
S_2 &= \mu\mathbf{t}.\textbf{Bob}?[\texttt{int}];\textbf{Bob}![\texttt{bool}];\mathbf{t}
\end{aligned}$$

They are syntactically different, but they expand to the same infinite tree:



So we can conclude S_1 is dual of S_2.

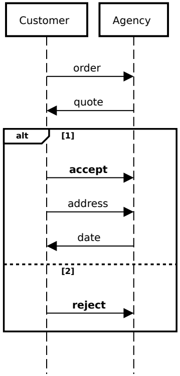
Quiz: Duality

1. Give a dual type for each (correct) type given in *Examples of Session Types*.
2. Give a dual type for each (correct) type given in *Examples of Recursive Types*.
3. Define a duality as a bisimilation relation and prove the duality between a pair of types you gave using the bisimilarity.

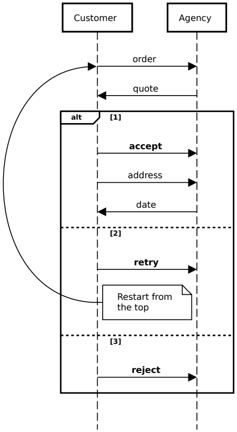
Recap: Two Travel Agencies

- Alice: a customer of the old agency.
- Bob: the clerk of the old agency.
- Charlie: a customer of the better agency.
- Eve: the clerk of the better agency.

Travel agency



Better Travel agency



Recap: Two Travel Agencies

Alice's interaction has the type:

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

Eve's has:

$$S_{\textbf{Bob}}{}' = \mu\mathbf{t}. \begin{array}{l} \textbf{Alice}?[\texttt{string}]; \textbf{Alice}![\texttt{int}]; \\ \textbf{Alice}\& \left\{ \begin{array}{l} \mathit{accept} : \textbf{Alice}?[\texttt{string}]; \\ \qquad\qquad \textbf{Alice}![\texttt{string}]; \texttt{end} \\ \mathit{reject} : \texttt{end} \\ \mathit{retry} : \mathbf{t} \end{array} \right\} \end{array}$$

Note that, the two types may not *not* dual of each other, and yet intuitively we know their communication does not lead to error.

Subtyping

We define a subtyping relation on session types.
Subtyping is defined *co*inductively to be the greatest relation satisfying the rules:

Subtyping rules

$$[\textsc{Sub-End}]\ \texttt{end} \leqslant \texttt{end}$$

$$[\textsc{Sub-Send}]\ \frac{S \leqslant S'}{\mathbf{p}![U]; S \leqslant \mathbf{p}![U]; S'}$$

$$[\textsc{Sub-Recv}]\ \frac{S \leqslant S'}{\mathbf{p}?[U]; S \leqslant \mathbf{p}?[U]; S'}$$

Subtyping rules (continued)

$$[\textsc{Sub-Bra}]\ \frac{\forall i \in I. S_i \leqslant S'_i}{\mathbf{p}\&\{l_i : S_i\}_{i\in I\cup J} \leqslant \mathbf{p}\&\{l_i : S'_i\}_{i\in I}}$$

Intuition: A process can implement more branches and forget about them.

$$[\textsc{Sub-Sel}]\ \frac{\forall i \in I. S_i \leqslant S'_i}{\mathbf{p}\oplus\{l_i : S_i\}_{i\in I} \leqslant \mathbf{p}\oplus\{l_i : S'_i\}_{i\in I\cup J}}$$

Intuition: A process can always make a choice in a wider range of choices.

Subtyping rules (continued)

$$[\textsc{Sub-}\mu\mathrm{L}]\ \frac{S[\mu\mathbf{t}.S/\mathbf{t}] \leqslant S'}{\mu\mathbf{t}.S \leqslant S'}$$

$$[\textsc{Sub-}\mu\mathrm{R}]\ \frac{S \leqslant S'[\mu\mathbf{t}.S'/\mathbf{t}]}{S \leqslant \mu\mathbf{t}.S'}$$

Two Travel Agencies Again

Alice has type:

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

Charlie has type:

$$S_{\textsf{Alice}}{}' = \mu \mathbf{t}. \begin{array}{l}\mathbf{Bob}![\texttt{string}];\mathbf{Bob}?[\texttt{int}];\\ \mathbf{Bob}\oplus\left\{\begin{array}{l} accept : \mathbf{Bob}![\texttt{string}]; \\ \qquad\qquad \mathbf{Bob}?[\texttt{string}];\mathsf{end} \\ reject : \mathsf{end} \\ retry : \mathbf{t}\end{array}\right\}\end{array}$$

So $S_{\textsf{Alice}} \leqslant S_{\textsf{Alice}}{}'$.

Two Travel Agencies Again

Bob has type:

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

Eve has type:

$$S_{\textbf{Bob}}{}' = \mu\mathbf{t}. \begin{array}{l} \textbf{Alice}?[\texttt{string}];\textbf{Alice}![\texttt{int}]; \\ \textbf{Alice}\&\left\{\begin{array}{l} \mathit{accept} : \textbf{Alice}?[\texttt{string}]; \\ \qquad\qquad \textbf{Alice}![\texttt{string}];\texttt{end} \\ \mathit{reject} : \texttt{end} \\ \mathit{retry} : \mathbf{t} \end{array}\right\} \end{array}$$

So $S_{\textbf{Bob}}{}' \leqslant S_{\textbf{Bob}}$.

The bigger picture

$$\begin{tikzcd} S_{\textsf{Alice}} \arrow[r, phantom, "\leqslant"] \arrow[d, leftrightarrow] & {S_{\textsf{Alice}}}' \arrow[d, leftrightarrow] \\ S_{\textsf{Bob}} \arrow[r, phantom, "\geqslant"] & {S_{\textsf{Bob}}}' \end{tikzcd}$$

1. Prove if $S_1 \leqslant S_2$ then $\overline{S_2} \leqslant \overline{S_1}$.
2. Give examples of subtyping to (3) and (6) in *Examples of Session Types*
3. (Challenging) Define a subtyping relation as a binary co-inductive relation $\mathcal{R} \subseteq (\mathcal{S} \times \mathcal{S})$ and use it to prove subtyping between a pair of types you gave.

Challenge Solution V1

Let $\mathcal{S}$ be a set of closed session types. A binary relation $\mathcal{R} \subseteq (\mathcal{S} \times \mathcal{S})$ is a *subtyping* whenever $S_1 \; \mathcal{R} \; S_2$ implies that:

- If $S_1 \xrightarrow{\mathbf{p}?[U]} S$ then $S_2 \xrightarrow{\mathbf{p}?[U]} S'$ with $S \; \mathcal{R} \; S'$
- If $S_1 \xrightarrow{\mathbf{p}![U]} S$ then $S_2 \xrightarrow{\mathbf{p}![U]} S'$ with $S \; \mathcal{R} \; S'$
- If $S_2 \xrightarrow{\mathbf{p}\&l} S'$ then $S_1 \xrightarrow{\mathbf{p}\&l} S$ with $S \; \mathcal{R} \; S'$
- If $S_1 \xrightarrow{\mathbf{p}\oplus l} S$ then $S_2 \xrightarrow{\mathbf{p}\oplus l} S'$ with $S \; \mathcal{R} \; S'$
- If $\forall \alpha$, $\nexists S$ such that $S_1 \xrightarrow{\alpha} S$ then $\forall \alpha', \nexists S'$ such that $S_2 \xrightarrow{\alpha'} S'$

Do the following hold? What about now?

$$\mathbf{end} \; \mathcal{R} \; \mathbf{p}?[U]; \mathbf{end}$$

$$\mu \mathbf{t}.\mathbf{end} \; \mathcal{R} \; \mathbf{end}$$

What is a potential issue in implementation?

Challenge Solution V2

Extending the semantics of the LTS allows us to distinguish reductions of terminal session types.

$$[\text{L-In}]\ \mathbf{end} \xrightarrow{\texttt{skip}} \texttt{skip}$$

Let $\mathcal{S}$ be a set of closed session types. A binary relation $\mathcal{R} \subseteq (\mathcal{S} \times \mathcal{S})$ is a *subtyping* whenever $S_1\ \mathcal{R}\ S_2$ implies that:

- If $S_1 \xrightarrow{\texttt{skip}} \texttt{skip}$ then $S_2 \xrightarrow{\texttt{skip}} \texttt{skip}$
- If $S_1 \xrightarrow{\mathbf{p}?[U]} S$ then $S_2 \xrightarrow{\mathbf{p}?[U]} S'$ with $S\ \mathcal{R}\ S'$
- If $S_1 \xrightarrow{\mathbf{p}![U]} S$ then $S_2 \xrightarrow{\mathbf{p}![U]} S'$ with $S\ \mathcal{R}\ S'$
- If $S_2 \xrightarrow{\mathbf{p}\&l} S'$ then $S_1 \xrightarrow{\mathbf{p}\&l} S$ with $S\ \mathcal{R}\ S'$
- If $S_1 \xrightarrow{\mathbf{p}\oplus l} S$ then $S_2 \xrightarrow{\mathbf{p}\oplus l} S'$ with $S\ \mathcal{R}\ S'$

Unfolding of Session Types

$$\begin{aligned} \mathtt{unf}(\mu\mathbf{t}.S) &= \mathtt{unf}(S[\mu\mathbf{t}.S/\mathbf{t}]) \\ \mathtt{unf}(S) &= S \text{ if } S \neq \mu\mathbf{t}.T \text{ for some } T \end{aligned}$$

Example:

$$\begin{aligned} \mathtt{unf}(\mu\mathbf{t}.\mathbf{p}![U];\mathbf{t}) &= \mathtt{unf}(\mathbf{p}![U];\mu\mathbf{t}.\mathbf{p}![U];\mathbf{t}) \\ &= \mathbf{p}![U];\mu\mathbf{t}.\mathbf{p}![U];\mathbf{t} \end{aligned}$$

Challenge Solution V3

Let $\mathcal{S}$ be a set of closed session types. A binary relation $\mathcal{R} \subseteq (\mathcal{S} \times \mathcal{S})$ is a *subtyping* whenever $S_1 \mathrel{\mathcal{R}} S_2$ implies that:

- If $\mathtt{unf}(S_1) = \mathtt{end}$ then $\mathtt{unf}(S_2) = \mathtt{end}$
- If $\mathtt{unf}(S_1) = \mathbf{p}![U]; S$ then $\mathtt{unf}(S_2) = \mathbf{p}![U]; S'$ with $S \mathrel{\mathcal{R}} S'$
- If $\mathtt{unf}(S_1) = \mathbf{p}?[U]; S$ then $\mathtt{unf}(S_2) = \mathbf{p}?[U]; S'$ with $S \mathrel{\mathcal{R}} S'$
- If $\mathtt{unf}(S_1) = \mathbf{p}\&\{l_i : S_i\}_{i \in I \cup J}$ then $\mathtt{unf}(S_2) = \mathbf{p}\&\{l_i : S'_i\}_{i \in I}$ and $\forall i \in I$ we have $S_i \mathrel{\mathcal{R}} S'_i$
- If $\mathtt{unf}(S_1) = \mathbf{p}\oplus\{l_i : S_i\}_{i \in I}$ then $\mathtt{unf}(S_2) = \mathbf{p}\oplus\{l_i : S'_i\}_{i \in I \cup J}$ and $\forall i \in I$ we have $S_i \mathrel{\mathcal{R}} S'_i$

Recap

Last week, we discussed about:

- Typing expressions
- The syntax of binary session types
- Recursive types
- The notion of duality
- Subtyping of session types

Typing Processes

In a context where we record typing assumptions, for processes we also need to store type variables.

$$\Gamma ::= \cdot \mid \Gamma, x : U \mid \Gamma, X : S$$

We assign session types to processes with a judgment:

$$\boxed{\Gamma \vdash P : S}$$

We read this judgment as:

Under typing context Γ, the process P has session type S.

Typing rules

$$[\textsc{Ty-End}]\ \frac{}{\Gamma \vdash \mathbf{0} : \mathtt{end}}$$

An inactive process $\mathbf{0}$ always has session type $\mathtt{end}$.

Typing rules

$$[\textsc{Ty-Send}]\ \frac{\Gamma \vdash e : U \qquad \Gamma \vdash P : S}{\Gamma \vdash \overline{\mathsf{p}}\,\langle e\rangle.P : \mathsf{p}![U];S}$$

A sending process $\overline{\mathsf{p}}\,\langle e\rangle.P$ has session type $\mathsf{p}![U];S$, if the expression e to send has sort U, and the process P has session type S.

$$[\textsc{Ty-Recv}]\ \frac{\Gamma, x : U \vdash P : S}{\Gamma \vdash \mathsf{p}\,(x).P : \mathsf{p}?[U];S}$$

A receiving process $\mathsf{p}\,(x).P$ has session type $\mathsf{p}?[U];S$, if the process P has session type S under the assumption that the variable x has sort U.

Example

$$\Gamma_1 = \mathit{address} : \mathtt{string}$$

$$\dfrac{\dfrac{}{\Gamma_1 \vdash \text{“20230815”} : \mathtt{string}} \qquad \dfrac{}{\Gamma_1 \vdash \mathbf{0} : \mathtt{end}}}{\dfrac{\Gamma_1 \vdash \overline{\textsf{Alice}}\langle \text{“20230815”} \rangle.\mathbf{0} : \textsf{Alice}![\mathtt{string}];\mathtt{end}}{\cdot \vdash \textsf{Alice}(\mathit{address}).\overline{\textsf{Alice}}\langle \text{“20230815”} \rangle.\mathbf{0} : \textsf{Alice}?[\mathtt{string}];\textsf{Alice}![\mathtt{string}];\mathtt{end}}}$$

Example

$$\Gamma_2 = \mathit{date} : \texttt{string}$$

$$\dfrac{\dfrac{}{\cdot \vdash \text{“}\texttt{L'Aquila}\text{”} : \texttt{string}} \qquad \dfrac{\dfrac{}{\Gamma_2 \vdash \mathbf{0} : \mathsf{end}}}{\cdot \vdash \mathsf{Bob}\,(\mathit{date}).\mathbf{0} : \mathsf{Bob}?[\texttt{string}];\mathsf{end}}}{\cdot \vdash \overline{\mathsf{Bob}}\,\langle\text{“}\texttt{L'Aquila}\text{”}\rangle.\mathsf{Bob}\,(\mathit{date}).\mathbf{0} : \mathsf{Bob}![\texttt{string}];\mathsf{Bob}?[\texttt{string}];\mathsf{end}}$$

Typing rules

$$[\textsc{Ty-Sel}]\ \frac{\Gamma \vdash P : S}{\Gamma \vdash \mathbf{p} \triangleleft l.P : \mathbf{p}\oplus\{l : S\}}$$

A selection process $\mathbf{p} \triangleleft l.P$ has session type $\mathbf{p}\oplus\{l : S\}$, if the process P has session type S.

$$[\textsc{Ty-Bra}]\ \frac{\forall j \in I . \Gamma \vdash P_j : S_j}{\Gamma \vdash \mathbf{p} \triangleright \{l_i : P_i\}_{i\in I} : \mathbf{p}\&\{l_i : S_i\}_{i\in I}}$$

A branching process $\mathbf{p} \triangleright \{l_i : P_i\}_{i\in I}$ has session type $\mathbf{p}\&\{l_i : S_i\}_{i\in I}$, if for all indices $i \in I$, the process P_i has session type S_i.

Example

$$\dfrac{\dfrac{}{\cdot \vdash \mathbf{0} : \mathtt{end}} \qquad \dfrac{\cdots}{\cdot \vdash \textsf{Alice}\,(address).\cdots : \textsf{Alice}?[\mathtt{string}];\cdots}}{\cdot \vdash \textsf{Alice} \triangleright \{reject : \mathbf{0}, accept : \cdots\} : \textsf{Alice}\&\{reject : \mathtt{end}, accept : \cdots\}}$$

Example

$$\dfrac{\dfrac{}{\cdot \vdash \mathbf{0} : \mathtt{end}}}{\cdot \vdash \textsf{Bob} \triangleleft \mathit{reject}.\mathbf{0} : \textsf{Bob}\oplus\{\mathit{reject} : \mathtt{end}\}}$$

$$\dfrac{\dfrac{\cdots}{\cdot \vdash \overline{\textsf{Bob}}\,\langle \text{“}\mathtt{L'Aquila}\text{”}\rangle.\cdots : \textsf{Bob}![\mathtt{string}];\cdots}}{\cdot \vdash \textsf{Bob} \triangleleft \mathit{accept}.\cdots : \textsf{Bob}\oplus\{\mathit{accept} : \cdots\}}$$

Typing rules

$$[\textsc{Ty-Sub}]\ \frac{\Gamma \vdash P : S \qquad S \leqslant S'}{\Gamma \vdash P : S'}$$

A process P has session type S' if it has session type S and S is a subtype of S'.

Recall that in $[\textsc{Ty-Sel}]$, the session type in the conclusion always has form $\mathbf{p}{\oplus}\{l : S\}$.
By composing $[\textsc{Ty-Sub}]$ and $[\textsc{Ty-Sel}]$, we can add more choices to the result type by subtyping.

Example

$$\dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathbf{0} : \mathtt{end}}}{\cdot \vdash \mathsf{Bob} \triangleleft \mathit{reject}.\mathbf{0} : \mathsf{Bob}\oplus\{\mathit{reject} : \mathtt{end}\}} \qquad \begin{array}{c} \mathsf{Bob}\oplus\{\mathit{reject} : \mathtt{end}\} \\ \leqslant \\ \mathsf{Bob}\oplus\left\{\begin{array}{l} \mathit{reject} : \mathtt{end} \\ \mathit{accept} : \cdots \end{array}\right\} \end{array}}{\cdot \vdash \mathsf{Bob} \triangleleft \mathit{reject}.\mathbf{0} : \mathsf{Bob}\oplus\{\mathit{reject} : \mathtt{end}, \mathit{accept} : \cdots\}}$$

$$\dfrac{\dfrac{\dfrac{\cdots}{\cdot \vdash \overline{\mathsf{Bob}}\,\langle\text{“}\mathtt{L'Aquila}\text{”}\rangle.\cdots : \mathsf{Bob}![\mathtt{string}];\cdots}}{\cdot \vdash \mathsf{Bob} \triangleleft \mathit{accept}.\cdots : \mathsf{Bob}\oplus\{\mathit{accept} : \cdots\}} \qquad \begin{array}{c} \mathsf{Bob}\oplus\{\mathit{accept} : \cdots\} \\ \leqslant \\ \mathsf{Bob}\oplus\left\{\begin{array}{l} \mathit{reject} : \mathtt{end} \\ \mathit{accept} : \cdots \end{array}\right\} \end{array}}{\cdot \vdash \mathsf{Bob} \triangleleft \mathit{accept}.\cdots : \mathsf{Bob}\oplus\{\mathit{reject} : \mathtt{end}, \mathit{accept} : \cdots\}}$$

Typing rules

$$[\textsc{Ty-If}]\ \frac{\Gamma \vdash e : \texttt{bool} \qquad \Gamma \vdash P : S \qquad \Gamma \vdash Q : S}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q : S}$$

A process $\texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q$ has session type S if the expression e has sort $\texttt{bool}$ and both P and Q have session type S.

Hint: If you have a derivation of P having session type S_1, and Q having session type S_2, you can try to use $[\textsc{Ty-Sub}]$ and find a type S such that $S_1 \leqslant S$ and $S_2 \leqslant S$.

Example

$$\dfrac{\begin{array}{c}\cdot \vdash \cdots : \texttt{bool} \\ \cdot \vdash \textsf{Bob} \triangleleft reject.\mathbf{0} : \textsf{Bob}\oplus\{reject : \texttt{end}, accept : \cdots\} \\ \cdot \vdash \textsf{Bob} \triangleleft accept.\cdots : \textsf{Bob}\oplus\{reject : \texttt{end}, accept : \cdots\}\end{array}}{\begin{array}{ll}\cdot \vdash \texttt{if}\ \cdots & : \textsf{Bob}\oplus\{reject : \texttt{end}, accept : \cdots\} \\ \quad \texttt{then}\ \textsf{Bob} \triangleleft reject.\mathbf{0} & \\ \quad \texttt{else}\ \textsf{Bob} \triangleleft accept.\cdots & \end{array}}$$

Typing rules

$$[\textsc{Ty-PVar}]\ \frac{}{\Gamma, X : S \vdash X : S}$$

A process X has session type S, if we know that information from the typing context.

$$[\textsc{Ty-Rec}]\ \frac{\Gamma, X : S \vdash P : S}{\Gamma \vdash \mu X.P : S}$$

A process $\mu X.P$ has session type S, if the process P has session type S under the assumption that X has session type S.
Hint: You may wish to use a recursive type for S here.

Example

Let's define:
$S_a = \mu \mathbf{t}.\textbf{Alice}![\texttt{int}]; \mathbf{t}$
And keep in mind that it is the same as:
$S_a = \textbf{Alice}![\texttt{int}]; \mu \mathbf{t}.\textbf{Alice}![\texttt{int}]; \mathbf{t}$

$$\dfrac{\dfrac{}{\cdot \vdash 7 : \texttt{int}} \qquad \dfrac{\dots}{\cdot \vdash \mu X.\overline{\textbf{Alice}}\langle 42 \rangle.X : S_a}}{\dfrac{\cdot \vdash \overline{\textbf{Alice}}\langle 7 \rangle.\mu X.\overline{\textbf{Alice}}\langle 42 \rangle.X : \textbf{Alice}![\texttt{int}]; S_a}{\cdot \vdash \overline{\textbf{Alice}}\langle 7 \rangle.\mu X.\overline{\textbf{Alice}}\langle 42 \rangle.X : S_a}}$$

Example

Remember:
$S_a = \mu \mathbf{t}.\mathsf{Alice}![\mathtt{int}];\mathbf{t}$

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\cdot \vdash 42 : \mathtt{int}} \qquad \cfrac{}{X : S_a \vdash X : S_a}}{X : S_a \vdash \overline{\mathsf{Alice}}\langle 42\rangle.X : \mathsf{Alice}![\mathtt{int}];S_a}}{X : S_a \vdash \overline{\mathsf{Alice}}\langle 42\rangle.X : S_a}}{\cdot \vdash \mu X.\overline{\mathsf{Alice}}\langle 42\rangle.X : S_a}$$

Composing processes

We use the judgment

$$\boxed{\vdash \mathcal{M}}$$

to say that $\mathcal{M}$ is well-typed.

The derivation rule is

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

which requires dual types for the two composed processes.

Examples

Are these $\mathcal{M}$ well-typed?

1. $\textbf{Alice} :: \overline{\textbf{Bob}}\,\langle 42\rangle.\mathbf{0} \mid$
   $\textbf{Bob} :: \textbf{Alice}\,(x).\texttt{if}\ x = 42\ \texttt{then}\ \mathbf{0}\ \texttt{else}\ \mathbf{0}$
2. $\textbf{Alice} :: \overline{\textbf{Bob}}\,\langle 42\rangle.\mathbf{0} \mid$
   $\textbf{Bob} :: \textbf{Alice}\,(x).\texttt{if}\ x = \text{“42”}\ \texttt{then}\ \mathbf{0}\ \texttt{else}\ \mathbf{0}$
3. $\textbf{Alice} :: \overline{\textbf{Bob}}\,\langle 42\rangle.\textbf{Bob}\,(y).\mathbf{0} \mid \textbf{Bob} :: \textbf{Alice}\,(x).\overline{\textbf{Alice}}\,\langle x+1\rangle.\mathbf{0}$
4. $\textbf{Alice} :: \overline{\textbf{Bob}}\,\langle 42\rangle.\mathbf{0} \mid \textbf{Bob} :: \textbf{Alice}\,(x).\overline{\textbf{Alice}}\,\langle x\rangle.\mathbf{0}$
5. $\textbf{Alice} :: \textbf{Bob} \triangleleft banana.\mathbf{0} \mid$
   $\textbf{Bob} :: \textbf{Alice} \triangleright \{apple : \mathbf{0}, banana : \mathbf{0}\}$
6. $\textbf{Alice} :: \textbf{Bob} \triangleleft orange.\mathbf{0} \mid$
   $\textbf{Bob} :: \textbf{Alice} \triangleright \{apple : \mathbf{0}, banana : \mathbf{0}\}$
7. $\textbf{Alice} :: \textbf{Bob} \triangleleft bye.\textbf{Bob} \triangleleft hello.\mathbf{0} \mid$
   $\textbf{Bob} :: \mu X.\textbf{Alice} \triangleright \{bye : \mathbf{0}, hello : X\}$

Examples

Is this well-typed?

$$\textsf{Alice} :: \texttt{if false then } \mathbf{0} \texttt{ else } \overline{\textsf{Bob}}\langle \texttt{"Hello"} \rangle.\mathbf{0} \mid \textsf{Bob} :: \textsf{Alice}(x).\mathbf{0}$$

$$\textsf{Alice} :: \texttt{if true then } \mathbf{0} \texttt{ else } \overline{\textsf{Bob}}\langle \texttt{"Hello"} \rangle.\mathbf{0} \mid \textsf{Bob} :: \textsf{Alice}(x).\mathbf{0}$$

Both are not well-typed, but the first will reduce to $\textsf{Alice} :: \mathbf{0} \mid \textsf{Bob} :: \mathbf{0}$, while the second will be stuck.

Preservation

Theorem (Preservation)

If $\mathcal{M}$ is well-typed (i.e. $\vdash \mathcal{M}$) and $\mathcal{M} \longrightarrow \mathcal{M}'$,
Then $\mathcal{M}'$ is well-typed (i.e. $\vdash \mathcal{M}'$).

Preservation is also known as *Subject Reduction*.

The presevation theorem states that well-typeness is *preserved* during reduction.

N.B. However, it doesn't mean that types of processes are preserved.

Progress

Theorem (Progress)

If $\mathcal{M}$ is well-typed (i.e. $\vdash \mathcal{M}$),
Then either there exists $\mathcal{M}'$ such that $\mathcal{M} \longrightarrow \mathcal{M}'$, or
$\mathcal{M} \equiv \textbf{Alice} :: \mathbf{0} \;\big|\; \textbf{Bob} :: \mathbf{0}$.

The progress theorem states that either a well-typed binary session has reached the end, or it can be further reduced. Implicitly, it states that a well-typed binary session does not get *stuck*.

Reading List for Session Types

To learn more about session types:

- A Very Gentle Introduction to Multiparty Session Types available on Materials
- On the Preciseness of Subtyping in Session Types
  DOI : 10.23638/LMCS-13(2:12)2017
  Logical Methods in Computer Science Vol. 13(2:12)2017, pp. 1—61

Kozen, D. and Silva, A. (2017). Practical coinduction. *Mathematical Structures in Computer Science*, 27(7):1132–1152.

Pierce, B. C. (2002). *Types and Programming Languages*. The MIT Press, 1st edition.