Binary Session Types

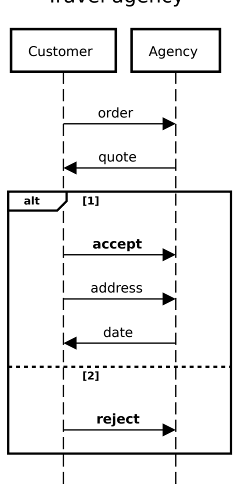
Francisco Ferreira Nobuko Yoshida Fangyi Zhou

Autumn 2019

Motivating Example: Travel Agency Example

This interaction can be described as:

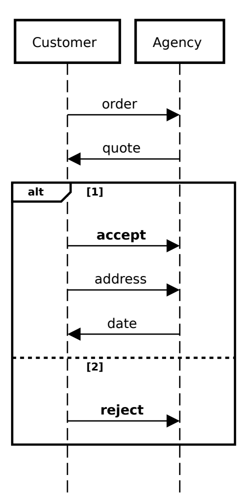
- The Customer posts an order



Motivating Example: Travel Agency Example

This interaction can be described as:

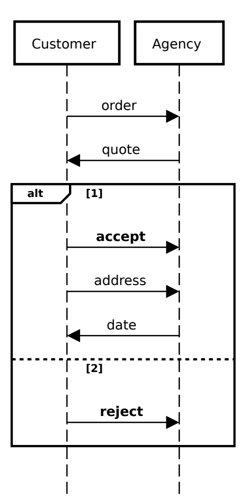
- The Customer posts an order
- The Agency sends a quote



Motivating Example: Travel Agency Example

This interaction can be described as:

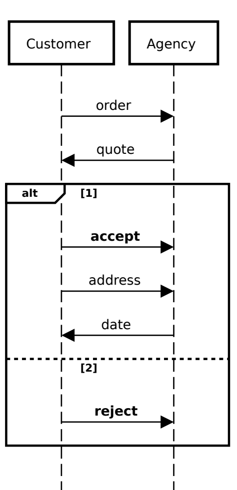
- The Customer posts an order
- The Agency sends a quote
- Then the customer can choose:



Motivating Example: Travel Agency Example

This interaction can be described as:

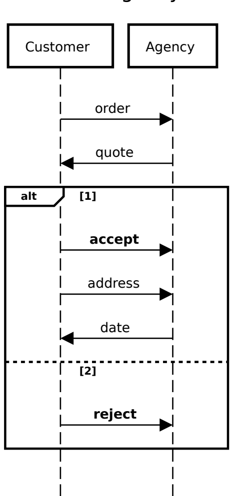
- The Customer posts an order
- The Agency sends a quote
- Then the customer can choose:
  - To accept the offer.



Motivating Example: Travel Agency Example

This interaction can be described as:

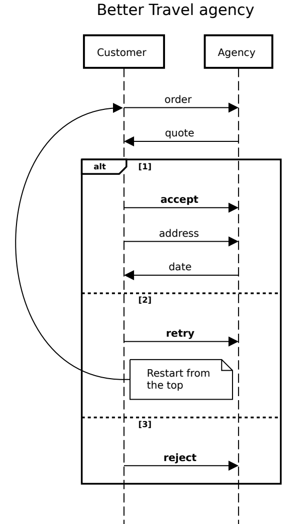
- The Customer posts an order
- The Agency sends a quote
- Then the customer can choose:
  - To `accept` the offer.
  - To simply `reject` the the quote.



The Better Travel Agency Example

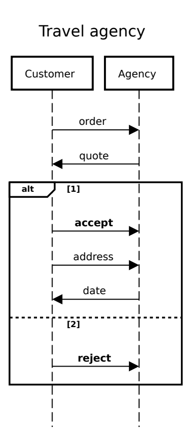
This interaction can be described as:

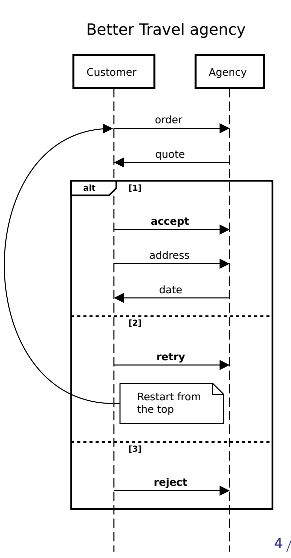
- The Customer posts an order
- The Agency sends a quote
- Then the customer can choose:
  - To `accept` the offer.
  - To `retry` with a new order (e.g.: to try a cheaper destination.)
  - To simply `reject` the the quote.



Some Sub-typing Intuition

Some of these processes seem to be related:

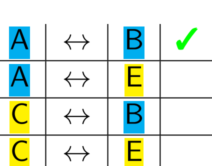




Some Sub-typing Intuition (2)

Dramatis Personæ:

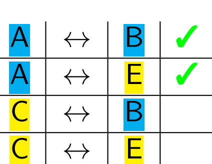
- Alice: a customer of the old agency.
- Bob: the clerk of the old agency.
- Charlie: a customer of the better agency.
- Eve: the clerk of the better agency.



Some Sub-typing Intuition (2)

Dramatis Personæ:

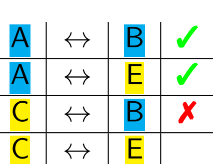
- Alice: a customer of the old agency.
- Bob: the clerk of the old agency.
- Charlie: a customer of the better agency.
- Eve: the clerk of the better agency.



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.

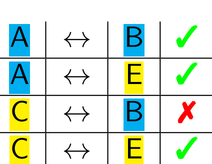


A	↔	B	✓
A	↔	E	✓
C	↔	B	✗
C	↔	E	

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.



Syntax of Expressions

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

Syntax of Expressions

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

$$
\begin{array}{rcll}
v & ::= & \underline{n} & \text{Integers} \\
 & \mid & \mathtt{true} \mid \mathtt{false} & \text{Booleans} \\
 & \mid & \text{``}\mathtt{str}\text{''} & \text{Strings} \\
e, e' & ::= & v & \text{Values}
\end{array}
$$

Syntax of Expressions

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

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

Syntax of Expressions

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

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

Syntax of Processes

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

Syntax of Processes

$\mathbf{p}$	$::=$	**Alice** $\mid$ **Bob**	Participant
P, Q	$::=$	$\mathbf{0}$	Inaction

Syntax of Processes

p	$::=$	**Alice** | **Bob**	Participant
P, Q	$::=$	$\mathbf{0}$	Inaction
	$\mid$	$\overline{\mathsf{p}}\,\langle e \rangle.P$	Message Send

Syntax of Processes

$\mathbf{p}$	$::=$	$\textbf{Alice} \mid \textbf{Bob}$	Participant
P, Q	$::=$	$\mathbf{0}$	Inaction
	$\mid$	$\overline{\mathbf{p}}\,\langle e \rangle.P$	Message Send
	$\mid$	$\mathbf{p}\,(x).P$	Message Receive

Syntax of Processes

$\mathbf{p}$	$::=$	$\textbf{Alice} \mid \textbf{Bob}$	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

Syntax of Processes

$$
\begin{array}{rcll}
\mathbf{p} & ::= & \mathbf{Alice} \mid \mathbf{Bob} & \text{Participant} \\
P, Q & ::= & \mathbf{0} & \text{Inaction} \\
& \mid & \overline{\mathbf{p}}\,\langle e \rangle.P & \text{Message Send} \\
& \mid & \mathbf{p}\,(x).P & \text{Message Receive} \\
& \mid & \mathbf{p} \triangleright \{l_i : P_i\}_{i \in I} & \text{Branching} \\
& \mid & \mathbf{p} \triangleleft l.P & \text{Selection}
\end{array}
$$

Syntax of Processes

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

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

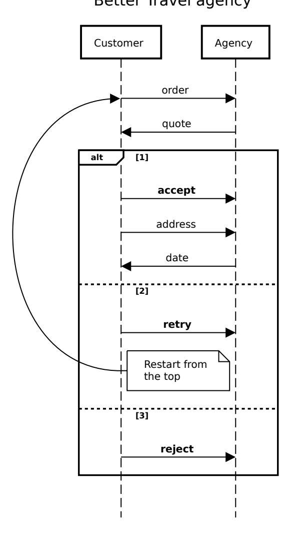
Syntax of Processes

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

Syntax of Processes

$\mathbf{p}$	$::=$	$\mathsf{Alice} \mid \mathsf{Bob}$	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

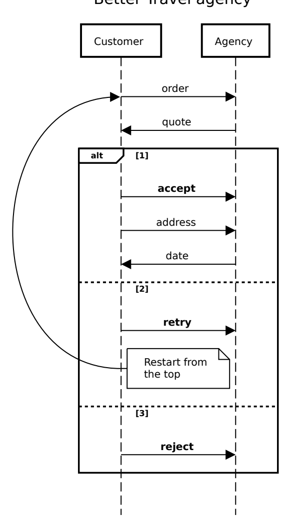
Travel Agency Revisited



For Customer (**Alice**):

$\overline{\textbf{Bob}}\langle\texttt{"Hawaii"}\rangle.$

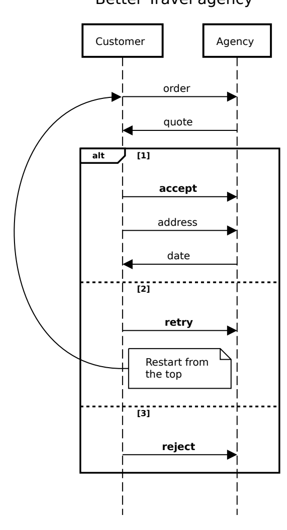
Travel Agency Revisited



For Customer (**Alice**):

$\overline{\textbf{Bob}}\,\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\textbf{Bob}\,(quote).$

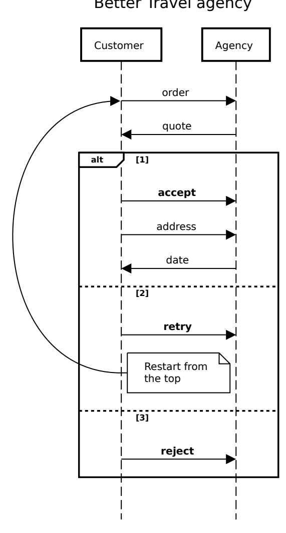
Travel Agency Revisited



For Customer (**Alice**):

$\overline{\textbf{Bob}}\langle\text{"Hawaii"}\rangle.\textbf{Bob}(quote).$
$\texttt{if}\ quote > 1000$
$\texttt{then}$

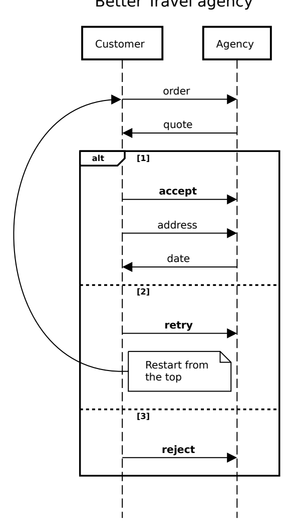
Travel Agency Revisited



For Customer (**Alice**):

$$
\begin{aligned}
&\overline{\textbf{Bob}}\,\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\textbf{Bob}\,(quote).\\
&\texttt{if } quote > 1000\\
&\texttt{then } \textbf{Bob} \triangleleft retry.\\
&\qquad \overline{\textbf{Bob}}\,\langle\text{“}\texttt{Brighton}\text{”}\rangle.\textbf{Bob}\,(newQuote).\\
&\qquad \texttt{if } newQuote > 1000\\
&\qquad \texttt{then } \textbf{Bob} \triangleleft reject.\mathbf{0}\\
&\qquad \texttt{else } \textbf{Bob} \triangleleft accept.\overline{\textbf{Bob}}\,\langle\text{“}\texttt{Woodward}\text{”}\rangle.\\
&\qquad\qquad \textbf{Bob}\,(date).\mathbf{0}\\
&\texttt{else}
\end{aligned}
$$

Travel Agency Revisited

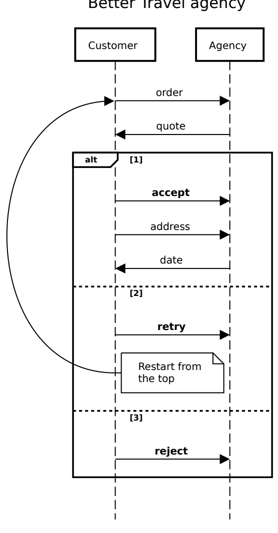


For Customer (**Alice**):

$$\overline{\textbf{Bob}}\langle\texttt{"Hawaii"}\rangle.\textbf{Bob}(quote).$$
$$\texttt{if } quote > 1000$$
$$\texttt{then } \textbf{Bob} \triangleleft retry.$$
$$\quad \overline{\textbf{Bob}}\langle\texttt{"Brighton"}\rangle.\textbf{Bob}(newQuote).$$
$$\quad \texttt{if } newQuote > 1000$$
$$\quad \texttt{then } \textbf{Bob} \triangleleft reject.\mathbf{0}$$
$$\quad \texttt{else } \textbf{Bob} \triangleleft accept.\overline{\textbf{Bob}}\langle\texttt{"Woodward"}\rangle.$$
$$\qquad \textbf{Bob}(date).\mathbf{0}$$
$$\texttt{else } \textbf{Bob} \triangleleft accept.\overline{\textbf{Bob}}\langle\texttt{"Woodward"}\rangle.$$
$$\quad \textbf{Bob}(date).\mathbf{0}$$

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

Travel Agency Revisited



For Agency (**Bob**):

$\mu X.$

Travel Agency Revisited

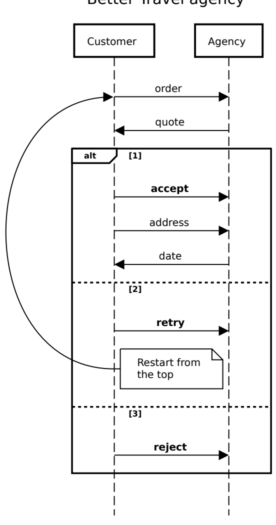
Better Travel agency

Customer
Agency
order
quote
alt [1]
accept
address
date
[2]
retry
Restart from the top
[3]
reject

For Agency (**Bob**):

$\mu X.$**Alice** $(order).$

Travel Agency Revisited



For Agency (**Bob**):

$$\mu X.\textbf{Alice}\,(order).\,\overline{\textbf{Alice}}\,\langle 1000 \oplus 5000 \rangle.$$

Travel Agency Revisited

For Agency (**Bob**):

$$\mu X.\mathbf{Alice}\,(order).\overline{\mathbf{Alice}}\,\langle 1000 \oplus 5000\rangle.\mathbf{Alice} \triangleright \begin{cases} retry : X \\ reject : \mathbf{0} \\ accept : \mathbf{Alice}\,(address).\overline{\mathbf{Alice}}\,\langle \text{“20191225”} \rangle.\mathbf{0} \end{cases}$$

We denote this process $P_{\mathbf{Bob}}$.

Composing Alice and Bob

In our calculus, we use

$$\textbf{Alice} :: P_{\textbf{Alice}} \mid \textbf{Bob} :: P_{\textbf{Bob}}$$

to compose the two processes in parallel.

Structural Congruence

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

$$\mu X.P \equiv P[\mu X.P/X] \quad [\text{C-Rec}]$$

Structural Congruence

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

$$\mu X.P \equiv P[\mu X.P/X] \quad [\text{C-Rec}]$$

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

Structural Congruence

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

$$\mu X.P \equiv P[\mu X.P/X] \quad [\text{C-Rec}]$$

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

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

The structural congruence relation is the smallest congruence relation containing the rules above.

[C-Rec] explained

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

$$\mu X.\overline{\textsf{Alice}}\langle 1\rangle.X$$

[C-Rec] explained

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

$$
\begin{aligned}
& \mu X.\overline{\textsf{Alice}}\langle 1\rangle.X \\
\equiv\ & \overline{\textsf{Alice}}\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\langle 1\rangle.X
\end{aligned}
$$

[C-Rec] explained

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

$$
\begin{aligned}
& \mu X.\overline{\textsf{Alice}}\,\langle 1\rangle.X \\
\equiv\ & \overline{\textsf{Alice}}\,\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\,\langle 1\rangle.X \\
\equiv\ & \overline{\textsf{Alice}}\,\langle 1\rangle.\overline{\textsf{Alice}}\,\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\,\langle 1\rangle.X
\end{aligned}
$$

[C-Rec] explained

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

$$
\begin{array}{ll}
 & \mu X.\overline{\textsf{Alice}}\langle 1\rangle.X \\
\equiv & \overline{\textsf{Alice}}\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\langle 1\rangle.X \\
\equiv & \overline{\textsf{Alice}}\langle 1\rangle.\overline{\textsf{Alice}}\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\langle 1\rangle.X \\
\equiv & \overline{\textsf{Alice}}\langle 1\rangle.\overline{\textsf{Alice}}\langle 1\rangle.\overline{\textsf{Alice}}\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\langle 1\rangle.X
\end{array}
$$

[C-Rec] explained

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

$$
\begin{array}{ll}
 & \mu X.\overline{\textsf{Alice}}\langle 1\rangle.X \\
\equiv & \overline{\textsf{Alice}}\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\langle 1\rangle.X \\
\equiv & \overline{\textsf{Alice}}\langle 1\rangle.\overline{\textsf{Alice}}\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\langle 1\rangle.X \\
\equiv & \overline{\textsf{Alice}}\langle 1\rangle.\overline{\textsf{Alice}}\langle 1\rangle.\overline{\textsf{Alice}}\langle 1\rangle.\mu X.\overline{\textsf{Alice}}\langle 1\rangle.X \\
\equiv & \cdots
\end{array}
$$

Expression Evaluation

Expression Evaluation

$$[\text{E-NonDet-L}]\ \frac{e_1 \downarrow v}{e_1 \oplus e_2 \downarrow v}$$

$$[\text{E-NonDet-R}]\ \frac{e_2 \downarrow v}{e_1 \oplus e_2 \downarrow v}$$

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

Operational Semantics

Operational Semantics

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

Operational Semantics

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

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

Operational Semantics

$$[\textsc{R-IfTrue}]\ \frac{e \downarrow \texttt{true}}{\texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q \longrightarrow P}$$

$$[\textsc{R-IfFalse}]\ \frac{e \downarrow \texttt{false}}{\texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q \longrightarrow Q}$$

Operational Semantics

$$[\text{R-IfTrue}]\ \frac{e \downarrow \texttt{true}}{\texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q \longrightarrow P}$$

$$[\text{R-IfFalse}]\ \frac{e \downarrow \texttt{false}}{\texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q \longrightarrow Q}$$

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

Travel Agency Revisited

$$\textbf{Alice} :: P_{\textbf{Alice}} \mid \textbf{Bob} :: P_{\textbf{Bob}}$$

Travel Agency Revisited

$$\begin{aligned} & \textsf{Alice} :: P_{\textsf{Alice}} \mid \textsf{Bob} :: P_{\textsf{Bob}} \\ = \quad & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \text{``}\texttt{Hawaii}\text{''}\rangle.\cdots \mid \textsf{Bob} :: \mu X.\textsf{Alice}(order).\cdots \end{aligned}$$

Travel Agency Revisited

$$
\begin{array}{rl}
 & \textsf{Alice} :: P_{\textsf{Alice}} \mid \textsf{Bob} :: P_{\textsf{Bob}} \\
= & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \text{“}\texttt{Hawaii}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \mu X.\textsf{Alice}(order).\cdots \\
\equiv & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \text{“}\texttt{Hawaii}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}(order).\cdots
\end{array}
$$

Travel Agency Revisited

$$
\begin{array}{rl}
& \textbf{Alice} :: P_{\textbf{Alice}} \mid \textbf{Bob} :: P_{\textbf{Bob}} \\
= & \textbf{Alice} :: \overline{\textbf{Bob}}\langle \texttt{"Hawaii"} \rangle.\cdots \mid \textbf{Bob} :: \mu X.\textbf{Alice}(order).\cdots \\
\equiv & \textbf{Alice} :: \overline{\textbf{Bob}}\langle \texttt{"Hawaii"} \rangle.\cdots \mid \textbf{Bob} :: \textbf{Alice}(order).\cdots \\
\longrightarrow & \textbf{Alice} :: \textbf{Bob}(quote).\cdots \mid \textbf{Bob} :: \overline{\textbf{Alice}}\langle 1000 \oplus 5000 \rangle.\cdots
\end{array}
$$

Travel Agency Revisited

$$
\begin{array}{ll}
 & \mathsf{Alice} :: P_{\mathsf{Alice}} \mid \mathsf{Bob} :: P_{\mathsf{Bob}} \\
= & \mathsf{Alice} :: \overline{\mathsf{Bob}}\langle \text{“}\mathtt{Hawaii}\text{”}\rangle.\cdots \mid \mathsf{Bob} :: \mu X.\mathsf{Alice}(order).\cdots \\
\equiv & \mathsf{Alice} :: \overline{\mathsf{Bob}}\langle \text{“}\mathtt{Hawaii}\text{”}\rangle.\cdots \mid \mathsf{Bob} :: \mathsf{Alice}(order).\cdots \\
\longrightarrow & \mathsf{Alice} :: \mathsf{Bob}(quote).\cdots \mid \mathsf{Bob} :: \overline{\mathsf{Alice}}\langle 1000 \oplus 5000\rangle.\cdots \\
\longrightarrow & \mathsf{Alice} :: \text{if } 1000 > 1000 \ \cdots \mid \mathsf{Bob} :: \mathsf{Alice} \triangleright \{\cdots\}
\end{array}
$$

Travel Agency Revisited

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

Travel Agency Revisited

$$
\begin{array}{ll}
 & \textsf{Alice} :: P_{\textsf{Alice}} \mid \textsf{Bob} :: P_{\textsf{Bob}} \\
= & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \texttt{"Hawaii"} \rangle.\cdots \mid \textsf{Bob} :: \mu X.\textsf{Alice}(order).\cdots \\
\equiv & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \texttt{"Hawaii"} \rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}(order).\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob}(quote).\cdots \mid \textsf{Bob} :: \overline{\textsf{Alice}}\langle 1000 \oplus 5000 \rangle.\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{if}\ 1000 > 1000\ \cdots \mid \textsf{Bob} :: \textsf{Alice} \triangleright \{\cdots\} \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob} \triangleleft accept.\cdots \mid \textsf{Bob} :: \textsf{Alice} \triangleright \{accept : \cdots ; \cdots\} \\
\longrightarrow & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \texttt{"Woodward"} \rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}(address).\cdots
\end{array}
$$

Travel Agency Revisited

$$
\begin{array}{rl}
 & \textsf{Alice} :: P_{\textsf{Alice}} \mid \textsf{Bob} :: P_{\textsf{Bob}} \\
= & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \text{“}\texttt{Hawaii}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \mu X.\textsf{Alice}(order).\cdots \\
\equiv & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \text{“}\texttt{Hawaii}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}(order).\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob}(quote).\cdots \mid \textsf{Bob} :: \overline{\textsf{Alice}}\langle 1000 \oplus 5000\rangle.\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{if}\ 1000 > 1000\ \cdots \mid \textsf{Bob} :: \textsf{Alice} \triangleright \{\cdots\} \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob} \triangleleft accept.\cdots \mid \textsf{Bob} :: \textsf{Alice} \triangleright \{accept : \cdots ; \cdots\} \\
\longrightarrow & \textsf{Alice} :: \overline{\textsf{Bob}}\langle \text{“}\texttt{Woodward}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}(address).\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob}(date).\mathbf{0} \mid \textsf{Bob} :: \overline{\textsf{Alice}}\langle \text{“}20191225\text{”}\rangle.\mathbf{0}
\end{array}
$$

Travel Agency Revisited

$$
\begin{array}{rl}
 & \textsf{Alice} :: P_{\textsf{Alice}} \mid \textsf{Bob} :: P_{\textsf{Bob}} \\
= & \textsf{Alice} :: \overline{\textsf{Bob}}\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \mu X.\textsf{Alice}(order).\cdots \\
\equiv & \textsf{Alice} :: \overline{\textsf{Bob}}\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}(order).\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob}(quote).\cdots \mid \textsf{Bob} :: \overline{\textsf{Alice}}\langle 1000 \oplus 5000\rangle.\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{if } 1000 > 1000 \ \cdots \mid \textsf{Bob} :: \textsf{Alice} \triangleright \{\cdots\} \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob} \triangleleft accept.\cdots \mid \textsf{Bob} :: \textsf{Alice} \triangleright \{accept : \cdots ; \cdots\} \\
\longrightarrow & \textsf{Alice} :: \overline{\textsf{Bob}}\langle\text{“}\texttt{Woodward}\text{”}\rangle.\cdots \mid \textsf{Bob} :: \textsf{Alice}(address).\cdots \\
\longrightarrow & \textsf{Alice} :: \textsf{Bob}(date).\mathbf{0} \mid \textsf{Bob} :: \overline{\textsf{Alice}}\langle\text{“}20191225\text{”}\rangle.\mathbf{0} \\
\longrightarrow & \textsf{Alice} :: \mathbf{0} \mid \textsf{Bob} :: \mathbf{0}
\end{array}
$$

Travel Agency Revisited

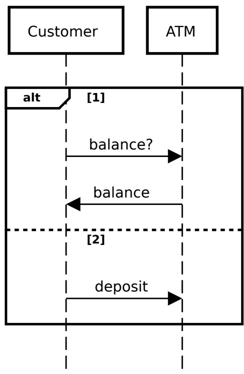
Try it yourself: How does the binary session reduce if

$$1000 \oplus 5000 \downarrow 5000$$

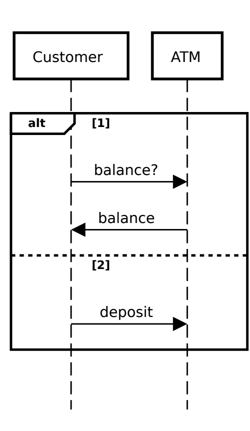
$$P_{\textsf{Alice}} = \begin{array}{l} \overline{\textsf{Bob}}\,\langle\text{“}\texttt{Hawaii}\text{”}\rangle.\textsf{Bob}\,(quote). \\ \texttt{if}\ quote > 1000 \\ \texttt{then}\ \textsf{Bob} \triangleleft retry.\overline{\textsf{Bob}}\,\langle\text{“}\texttt{Brighton}\text{”}\rangle.\textsf{Bob}\,(newQuote).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

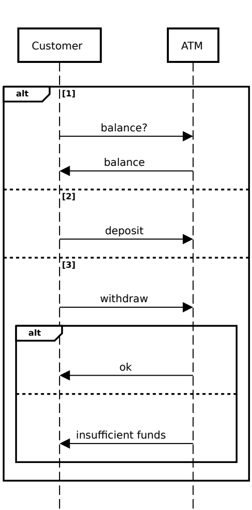
$$P_{\textsf{Bob}} = \begin{array}{l} \mu X.\textsf{Alice}\,(order).\overline{\textsf{Alice}}\,\langle 1000 \oplus 5000\rangle. \\ \qquad \textsf{Alice} \triangleright \left\{ \begin{array}{l} retry : X \\ reject : \mathbf{0} \\ accept : \textsf{Alice}\,(address).\overline{\textsf{Alice}}\,\langle\text{“}20191225\text{”}\rangle.\mathbf{0} \end{array} \right\} \end{array}$$

Extra example: to discuss

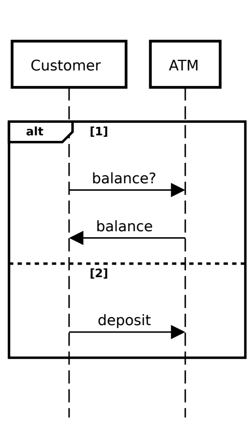


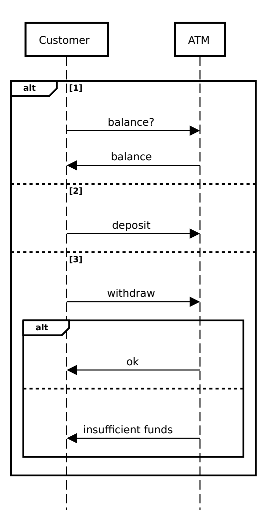
Extra example: to discuss

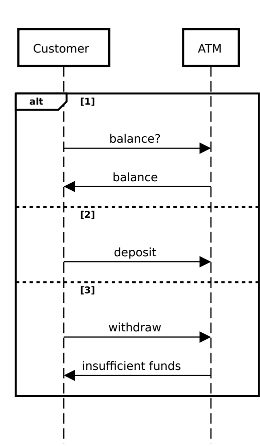




Extra example: to discuss







Recap

Last week, we discussed about:

- The syntax of binary session calculus

Recap

Last week, we discussed about:

- The syntax of binary session calculus
- The operational semantics

Recap

Last week, we discussed about:

- The syntax of binary session calculus
- The operational semantics
- Some examples

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:

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

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

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

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

Inference rules

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

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

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

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: $A \clubsuit B$

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

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:

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

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.[1]

[1] One may prove the latter property formally.

Typing rules define the judgment

$$[\textsc{Ty-Int}]\ \frac{}{\Gamma \vdash \underline{n} : \texttt{int}}$$

Typing rules define the judgment

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

Typing rules define the judgment

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

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

Typing rules define the judgment

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

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

$$[\textsc{Ty-Not}]\ \frac{\Gamma \vdash e : \texttt{bool}}{\Gamma \vdash \neg e : \texttt{bool}}$$

Typing rules define the judgment

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

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

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

Typing rules define the judgment

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

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

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

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

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`.

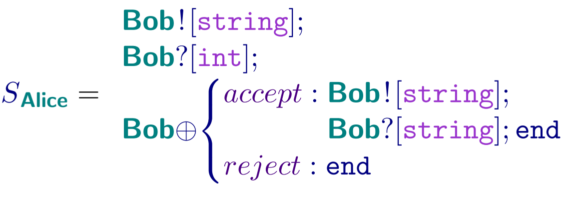
Examples of Session Types

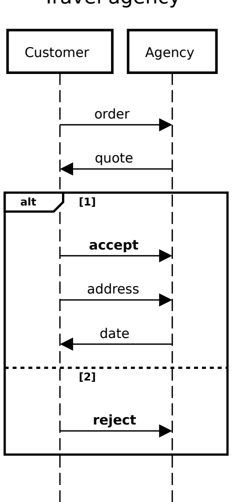
1. $\textbf{Alice}![\texttt{int}];\textbf{Alice}![\texttt{string}];\textbf{Alice}?[\texttt{int}];\textbf{end}$
2. $\textbf{Alice}![\texttt{int}];\textbf{Alice}![\texttt{string}];\textbf{Alice}?[\texttt{int}];$
3. $\textbf{Bob}\&\left\{\begin{array}{l} orange : \textbf{Bob}![\texttt{string}];\textbf{Bob}![\texttt{int}];\textbf{end} \\ cherry : \textbf{Bob}?[\texttt{string}];\textbf{end} \\ reject : \textbf{end} \end{array}\right\}$
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?
   $\textbf{Alice}\oplus\left\{\begin{array}{l} orange : \textbf{Alice}![\texttt{string}];\textbf{Alice}![\texttt{int}];\mathbf{t} \\ cherry : \textbf{Alice}?[\texttt{string}];\textbf{end} \\ repeat : \mathbf{t} \end{array}\right\}$

Travel Agency Revisited

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

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

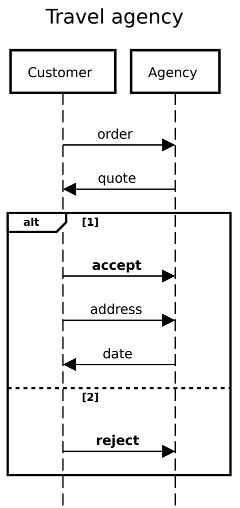




Travel Agency Revisited

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

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



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.

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. $\mu \mathbf{t}.\textsf{Alice}?[\mathbf{t}];$

Duality of Binary Session Types

We first define Duality of participants:

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

Duality of Binary Session Types

Then we define duality of binary session types *co*inductively.

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

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

Travel Agency Revisited

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

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

Are they dual of each other?

Duality, Informally

Duality provides a notion of compatibility.

Intuitively, two processes with dual types can communicate without error.[2]

[2] Yet we will not prove it formally in this course

Coinduction in Duality

Let

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

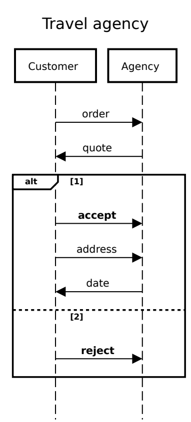
Is S_1 dual of S_2?

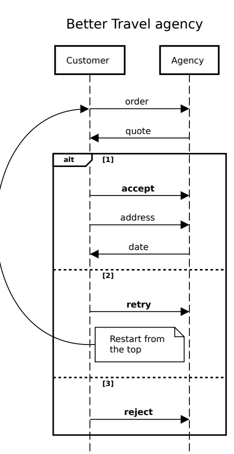
Quiz: Duality

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*.

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.





Recap: Two Travel Agencies

Alice's interaction has the type:

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

Eve's has:

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

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

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

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

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

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.

Two Travel Agencies Again

Alice has type:

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

Charlie has type:

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

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

Two Travel Agencies Again

Bob has type:

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

Eve has type:

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

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

The bigger picture

$$
\begin{array}{ccc}
S_{\textbf{Alice}} & \leqslant & S_{\textbf{Alice}}' \\
\updownarrow & & \updownarrow \\
S_{\textbf{Bob}} & \geqslant & S_{\textbf{Bob}}'
\end{array}
$$

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*

Recap

Last week, we discussed about:

- The syntax of binary session types

Recap

Last week, we discussed about:

- The syntax of binary session types
- The notion of duality

Recap

Last week, we discussed about:

- The syntax of binary session types
- The notion of duality
- Subtyping of session types

Recap

Last week, we discussed about:

- The syntax of binary session 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.

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 expression variable e has sort U.

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 expression variable e has sort U.

Example

$$\frac{}{\cdot \vdash \textbf{Alice}\,(address).\cdots : \textbf{Alice}?[\texttt{string}];\cdots}$$

Example

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

$$\dfrac{\overline{\Gamma_1 \vdash \overline{\textsf{Alice}}\langle \text{“20191225”} \rangle.\mathbf{0} : \textsf{Alice}![\mathtt{string}]; \mathtt{end}}}{\cdot \vdash \textsf{Alice}\,(address).\cdots : \textsf{Alice}?[\mathtt{string}]; \cdots}$$

Example

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

$$\dfrac{\dfrac{\overline{\Gamma_1 \vdash \text{“20191225”} : \mathtt{string}}}{\Gamma_1 \vdash \overline{\mathsf{Alice}}\langle \text{“20191225”} \rangle.\mathbf{0} : \mathsf{Alice}![\mathtt{string}]; \mathtt{end}}}{\cdot \vdash \mathsf{Alice}(address).\cdots : \mathsf{Alice}?[\mathtt{string}]; \cdots}$$

Example

$$\Gamma_1 = address : \texttt{string}$$

$$\dfrac{\dfrac{}{\Gamma_1 \vdash \text{“20191225”} : \texttt{string}} \qquad \dfrac{}{\Gamma_1 \vdash \mathbf{0} : \texttt{end}}}{\dfrac{\Gamma_1 \vdash \overline{\textsf{Alice}}\langle\text{“20191225”}\rangle.\mathbf{0} : \textsf{Alice}![\texttt{string}]; \texttt{end}}{\cdot \vdash \textsf{Alice}(address).\cdots : \textsf{Alice}?[\texttt{string}]; \cdots}}$$

Example

$$\frac{}{\cdot \vdash \overline{\textbf{Bob}}\langle\text{"Woodward"}\rangle.\cdots : \textbf{Bob}![\texttt{string}];\cdots}$$

Example

$$\frac{\dfrac{}{\cdot \vdash \text{“Woodward”} : \mathtt{string}}}{\cdot \vdash \overline{\mathbf{Bob}}\langle\text{“Woodward”}\rangle.\cdots : \mathbf{Bob}![\mathtt{string}];\cdots}$$

Example

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

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

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.

Typing rules

$$[\text{TY-SEL}]\ \frac{\Gamma \vdash P : S}{\Gamma \vdash \mathbf{p} \triangleleft l.P : \mathbf{p}\oplus\{l : S\}}$$

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

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

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

$$\frac{}{\cdot \vdash \textsf{Alice} \triangleright \{reject : \mathbf{0}, accept : \cdots\} : \textsf{Alice}\&\{reject : \texttt{end}, accept : \cdots\}}$$

Example

$$\overline{\cdot \vdash \textsf{Alice} \triangleright \{reject : \mathbf{0}, accept : \cdots\} : \textsf{Alice} \& \{reject : \texttt{end}, accept : \cdots\}}$$

Example

$$\dfrac{\dfrac{}{\cdot \vdash \mathbf{0} : \mathtt{end}}}{\cdot \vdash \textsf{Alice} \triangleright \{reject : \mathbf{0}, accept : \cdots\} : \textsf{Alice} \& \{reject : \mathtt{end}, accept : \cdots\}}$$

Example

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

Example

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

Example

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

Example

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

$$\dfrac{}{\cdot \vdash \textbf{Bob} \triangleleft \mathit{accept}.\cdots : \textbf{Bob} \oplus \{\mathit{accept} : \cdots\}}$$

Example

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

$$\dfrac{\dfrac{\cdots}{\cdot \vdash \overline{\textbf{Bob}}\langle\text{“}\mathtt{Woodward}\text{”}\rangle.\cdots : \textbf{Bob}![\mathtt{string}];\cdots}}{\cdot \vdash \textbf{Bob} \triangleleft \mathit{accept}.\cdots : \textbf{Bob}\oplus\{\mathit{accept} : \cdots\}}$$

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'.

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

$$\frac{}{\cdot \vdash \mathbf{Bob} \triangleleft \mathit{reject}.\mathbf{0} : \mathbf{Bob} \oplus \{\mathit{reject} : \mathtt{end}, \mathit{accept} : \cdots\}}$$

Example

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

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

Example

$$\dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathbf{0} : \mathsf{end}}}{\cdot \vdash \mathbf{Bob} \triangleleft \mathit{reject}.\mathbf{0} : \mathbf{Bob} \oplus \{\mathit{reject} : \mathsf{end}\}} \quad \begin{array}{c} \mathbf{Bob} \oplus \{\mathit{reject} : \mathsf{end}\} \\ \leqslant \\ \mathbf{Bob} \oplus \begin{Bmatrix} \mathit{reject} : \mathsf{end} \\ \mathit{accept} : \cdots \end{Bmatrix} \end{array}}{\cdot \vdash \mathbf{Bob} \triangleleft \mathit{reject}.\mathbf{0} : \mathbf{Bob} \oplus \{\mathit{reject} : \mathsf{end}, \mathit{accept} : \cdots\}}$$

Example

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

$$\overline{\cdot \vdash \mathsf{Bob} \triangleleft \mathit{reject}.\mathbf{0} : \mathsf{Bob}\oplus\{\mathit{reject} : \mathsf{end}, \mathit{accept} : \cdots\}}$$

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

$$\overline{\cdot \vdash \mathsf{Bob} \triangleleft \mathit{accept}.\cdots : \mathsf{Bob}\oplus\{\mathit{reject} : \mathsf{end}, \mathit{accept} : \cdots\}}$$

Typing rules

$$[\text{TY-IF}]\ \frac{\Gamma \vdash e : \texttt{bool} \qquad \Gamma \vdash P : S \qquad \Gamma \vdash Q : S}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q : S}$$

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.

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

$$\frac{}{\cdot \vdash \texttt{if} \cdots \quad \texttt{then } \textbf{Bob} \triangleleft \mathit{reject}.\mathbf{0} \quad \texttt{else } \textbf{Bob} \triangleleft \mathit{accept}.\cdots \; : \textbf{Bob} \oplus \{\mathit{reject} : \texttt{end}, \mathit{accept} : \cdots\}}$$

Example

$$\frac{\cdot \vdash \cdots : \texttt{bool}}{\cdot \vdash \texttt{if} \cdots \; \texttt{then} \; \textbf{Bob} \triangleleft \mathit{reject}.\mathbf{0} \; \texttt{else} \; \textbf{Bob} \triangleleft \mathit{accept}.\cdots : \textbf{Bob} \oplus \{\mathit{reject} : \textbf{end}, \mathit{accept} : \cdots\}}$$

Example

$$\dfrac{\cdot \vdash \cdots : \texttt{bool} \qquad \cdot \vdash \textbf{Bob} \triangleleft \mathit{reject}.\mathbf{0} : \textbf{Bob}\oplus\{\mathit{reject} : \textbf{end}, \mathit{accept} : \cdots\}}{\cdot \vdash \texttt{if}\ \cdots\ \texttt{then}\ \textbf{Bob} \triangleleft \mathit{reject}.\mathbf{0}\ \texttt{else}\ \textbf{Bob} \triangleleft \mathit{accept}.\cdots : \textbf{Bob}\oplus\{\mathit{reject} : \textbf{end}, \mathit{accept} : \cdots\}}$$

Example

$$\frac{\cdot \vdash \cdots : \texttt{bool} \quad \cdot \vdash \textbf{Bob} \lhd reject.\mathbf{0} : \textbf{Bob}\oplus\{reject : \texttt{end}, accept : \cdots\} \quad \cdot \vdash \textbf{Bob} \lhd accept.\cdots : \textbf{Bob}\oplus\{reject : \texttt{end}, accept : \cdots\}}{\cdot \vdash \texttt{if} \cdots \ \texttt{then}\ \textbf{Bob} \lhd reject.\mathbf{0}\ \texttt{else}\ \textbf{Bob} \lhd accept.\cdots : \textbf{Bob}\oplus\{reject : \texttt{end}, accept : \cdots\}}$$

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.

Typing rules

$$[\text{TY-PVAR}]\ \frac{}{\Gamma, X : S \vdash X : S}$$

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

$$[\text{TY-REC}]\ \frac{\Gamma, X : S \vdash P : S}{\Gamma \vdash \mu X.P : S}$$

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

Example

Let's define:

$S_a = \mu \mathbf{t}.\textsf{Alice}![\texttt{int}];\mathbf{t}$

And keep in mind that it is the same as:

$S_a = \textsf{Alice}![\texttt{int}];\mu \mathbf{t}.\textsf{Alice}![\texttt{int}];\mathbf{t}$

Example

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

$$\dfrac{}{\cdot \vdash \overline{\textsf{Alice}}\,\langle 7\rangle.\mu X.\overline{\textsf{Alice}}\,\langle 42\rangle.X \,:\, S_a}$$

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

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

Example

Let's define:

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

And keep in mind that it is the same as:

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

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

Example

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

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

Example

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

$$\frac{}{\cdot \vdash \mu X.\overline{\textsf{Alice}}\langle 42\rangle.X : S_a}$$

Example

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

$$\dfrac{\dfrac{}{X : S_a \vdash \overline{\textsf{Alice}}\langle 42 \rangle.X : S_a}}{\cdot \vdash \mu X.\overline{\textsf{Alice}}\langle 42 \rangle.X : S_a}$$

Example

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

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

Example

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

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

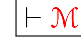
Example

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

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

Composing processes

We use the judgment



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

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

Examples

Is this well-typed?

$$\textbf{Alice} :: \texttt{if false then } \mathbf{0} \texttt{ else } \overline{\textbf{Bob}}\langle \text{“}\texttt{Hello}\text{”}\rangle.\mathbf{0} \mid \textbf{Bob} :: \textbf{Alice}(x).\mathbf{0}$$

Examples

Is this well-typed?

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

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

Examples

Is this well-typed?

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

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

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

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.

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*
$\mathcal{M} \equiv \textbf{Alice} :: \mathbf{0} \mid \textbf{Bob} :: \mathbf{0}$.

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
$\mathcal{M} \equiv \textbf{Alice} :: \mathbf{0} \mid \textbf{Bob} :: \mathbf{0}$.

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.