Imperial College
London

NOTES

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Session Types

Author:
W (CID: your college-id number)

Date: November 21, 2019

1 Binary Session Types

In this chapter we only focus on binary session types, i.e. the communication between two participants.

1.1 Expressions

We first introduce a simple Expression language:

$$
\begin{array}{rr}
v ::= \underline{n} & \text{Integers} \\
\mid true \mid false & \text{Booleans} \\
\mid \text{"}str\text{"} & \text{Strings} \\
e, e' ::= v & \text{Values} \\
\mid x & \text{Variables} \\
\mid e + e' \mid e - e' \mid -e & \text{Arithmetics} \\
\mid e = e' \mid e < e' \mid e > e' & \text{Relations} \\
\mid e \wedge e' \mid e \vee e' \mid \neg e & \text{Logical} \\
\mid e \oplus e' & \text{Non-determinism}
\end{array}
$$

1.1.1 Expression Evaluations

Non-determinisms

$$\frac{e_1 \downarrow v}{e_1 \oplus e_2 \downarrow v} \qquad \frac{e_2 \downarrow v}{e_1 \oplus e_2 \downarrow v}$$

Communication

$$\frac{e \downarrow v \quad \boldsymbol{p} \neq \boldsymbol{q}}{\boldsymbol{p} :: \bar{\boldsymbol{q}}\langle e \rangle.P \mid \boldsymbol{q} :: \boldsymbol{p}(x).Q \rightarrow \boldsymbol{p} :: P \mid \boldsymbol{q} :: Q[v/x]}$$

Branching and Selection

$$\frac{\exists j \in I_j = l \quad \boldsymbol{p} \neq \boldsymbol{q}}{\boldsymbol{p} :: \boldsymbol{q} \triangleleft l.P \mid \boldsymbol{q} :: \boldsymbol{p} \triangleright \{l_i : Q_i\}_{i \in I} \rightarrow \boldsymbol{p} :: P \mid \boldsymbol{q} :: Q_j}$$

Conditionals

$$\frac{e \downarrow true}{if\ e\ then\ P\ else\ Q \rightarrow P} \qquad \frac{e \downarrow false}{if\ e\ then\ P\ else\ Q \rightarrow Q}$$

Congruent

$$\frac{\mathcal{M}_1 \equiv \mathcal{M}_1' \quad \mathcal{M}_1' \rightarrow \mathcal{M}_2' \quad \mathcal{M}_2' \equiv \mathcal{M}_2}{\mathcal{M}_1 \rightarrow \mathcal{M}_2}$$

1.2 Processes

$$
\begin{array}{rr}
\boldsymbol{p} ::= \boldsymbol{Alice} \mid \boldsymbol{Bob} & \text{Participants} \\
P, Q ::= \mathbf{0} & \text{Inaction} \\
\mid \bar{\boldsymbol{p}}\langle e \rangle.P & \text{Send Message} \\
\mid \boldsymbol{p}(x).P & \text{Receive Message} \\
\mid \boldsymbol{p} \triangleright \{l_i : P_i\}_{i \in I} & \text{Branching} \\
\mid \boldsymbol{p} \triangleleft l.P & \text{Selection} \\
\mid if\ e\ then\ P\ else\ Q & \text{Conditional} \\
\mid \mu X.P & \text{Recursive Processes} \\
\mid X & \text{Process Variable} \\
\mathcal{M} ::= \boldsymbol{p} :: P \mid \boldsymbol{q} :: Q & \text{Binary Composition}
\end{array}
$$

The semantics for processes is a little bit different from π-calculus. In this example, we define two participants $\boldsymbol{Alice}$ and $\boldsymbol{Bob}$. The following process means Alice sends a message to Bob, and Bob receives a message from Alice.

$$\boldsymbol{Alice} :: \overline{\boldsymbol{Bob}}\langle "message" \rangle.P \mid \boldsymbol{Bob} :: \boldsymbol{Alice}(msg).P$$

1.2.1 Structural Congruence

$$
\begin{aligned}
&\mu X.P \equiv P[\mu X.P/X] \\
&\boldsymbol{p} :: P \mid \boldsymbol{q} :: Q \equiv \boldsymbol{q} :: Q \mid \boldsymbol{p} :: P \\
&P \equiv P' \Rightarrow \boldsymbol{p} :: P \mid \boldsymbol{q} :: Q \equiv \boldsymbol{p} :: P' \mid \boldsymbol{q} :: Q
\end{aligned}
$$

1.3 Type Systems

We can assign **Sorts** to the expressions defined above

$$U ::= int \mid bool \mid string$$

A **Typing context** stores information about variables and their sorts.

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

We assign sorts to expressions using a **judgement**, $\Gamma \vdash e : U$. Under a typing context Γ, expression e has sort U. The typing rules are defined as follow

$$\frac{}{\Gamma \vdash \underline{n} : int} \qquad TY - Int$$

$$\frac{}{\Gamma,\ x : U \vdash x : U} \qquad TY - Var$$

$$\frac{\Gamma \vdash e : int \quad \Gamma \vdash e' : int}{\Gamma \vdash e + e' : int} \qquad TY - Plus$$

$$\frac{\Gamma \vdash e : int \quad \Gamma \vdash e' : int}{\Gamma \vdash e < e' : bool} \qquad TY - Less$$

$$\frac{\Gamma \vdash e : bool}{\Gamma \vdash \neg e : bool} \qquad TY - Not$$

$$\frac{\Gamma \vdash e : U \quad \Gamma \vdash e' : U}{\Gamma \vdash e \oplus e' : U} \qquad TY - NonDet$$

1.4 Session Types

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

We define a session type for each participant, meaning a session type can only contain one participant. The below example is a session type for ***Bob***. This means send an int to ***Alice*** then receive a string from ***Alice*** then give ***Alice*** a choice.

$$\boldsymbol{Alice}![int];\boldsymbol{Alice}?[string];\boldsymbol{Alice}\&\{accept : \boldsymbol{Alice}![int],\ reject : end\}$$

From ***Alice*** perspective, the corresponding session type is

$$\boldsymbol{Bob}?[int];\boldsymbol{Bob}![string];\boldsymbol{Bob} \oplus \{accept : \boldsymbol{Bob}?[int],\ reject : end\}$$

Recursive Type is similar to Recursive process where $\mu \boldsymbol{t}.S \equiv S\{\mu \boldsymbol{t}.S/\boldsymbol{t}\}$. We assume types are closed and guarded and so $\mu \boldsymbol{t}.\boldsymbol{t}$ is not allowed.

1.5 Duality

In this chapter we only consider binary session types containing only two participants: ***Alice*** and ***Bob***. We define Duality of participants as:

$$\boldsymbol{Alice}\dagger = \boldsymbol{Bob} \qquad\qquad \boldsymbol{Bob}\dagger = \boldsymbol{Alice}$$

For the two participants, we can define duality for binary session types given $\boldsymbol{p}\dagger = \boldsymbol{q}$

$$\overline{end} = end$$

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

Two processes with dual types can communicate without error.

1.6 Subtyping

Consider the following session types

$$S_{\boldsymbol{Alice}} = \boldsymbol{Bob} \oplus \begin{cases} option1: \ \boldsymbol{Bob}![string] \\ option2: \ \boldsymbol{Bob}?[string] \end{cases} \qquad S_{\boldsymbol{Bob}} = \boldsymbol{Alice}\& \begin{cases} option1: \ \boldsymbol{Alice}?[string] \\ option2: \ \boldsymbol{Alice}![string] \end{cases}$$

$$S_{\boldsymbol{Alice'}} = \boldsymbol{Bob'} \oplus \begin{cases} option1: \ \boldsymbol{Bob}![string] \\ option2: \ \boldsymbol{Bob}?[string] \\ option3: \ \boldsymbol{Bob}?[int] \end{cases} \qquad S_{\boldsymbol{Bob'}} = \boldsymbol{Alice'}\& \begin{cases} option1: \ \boldsymbol{Alice}?[string] \\ option2: \ \boldsymbol{Alice}![string] \\ option3: \ \boldsymbol{Alice}![int] \end{cases}$$

We know that $\boldsymbol{Alice}$ and $\boldsymbol{Bob}$ can communicate without errors because they are dual type, similarly $\boldsymbol{Alice'}$ and $\boldsymbol{Bob'}$ can communicate without errors.

Intuitively, we know that $\boldsymbol{Alice}$ and $\boldsymbol{Bob'}$ should also be able to communicate without errors although they are not dual types, because $\boldsymbol{Alice}$ made a selection within the options $\boldsymbol{Bob'}$ provided.

In contrast, $\boldsymbol{Alice'}$ cannot communicate with $\boldsymbol{Bob}$ because an error will occur if $\boldsymbol{Alice'}$ decided to select $option3$.

We define subtyping relations as follow:

$$end \leqslant end \qquad Sub-End$$

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

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

$$\frac{\forall i \in I.\ S_i \leqslant S_i'}{\boldsymbol{p}\&\{I_i : S_i\}_{i \in I \cup J} \leqslant \boldsymbol{p}\&\{l_i : S_i'\}_{i \in I}} \qquad Sub-Bra$$

$$\frac{\forall i \in I,\ S_i \leqslant S_i'}{\boldsymbol{p} \oplus \{l_i : S_i\}_{i \in I} \leqslant \boldsymbol{p} \oplus \{l_i : S_i'\}_{i \in I \cup J}} \quad Sub - Sel$$

The intuition is that you can try to remove branches from branching to make them a dual of another process, or we can add more selections to a process to make them a dual of anther process.

Using the example above, $S_{\boldsymbol{Bob'}} \leqslant S_{\boldsymbol{Bob}}$ and $S_{\boldsymbol{Alice}} \leqslant S_{\boldsymbol{Alice'}}$.

$\boldsymbol{Alice}$ can communicate with $\boldsymbol{Bob'}$ because $S_{\boldsymbol{Alice}} = \overline{S_{\boldsymbol{Bob}}}$

1.7 Typing Processes

In this section, we extend the typing context to assign session types to processes.

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

We assign session types to processes using a judgement, $\Gamma \vdash X : S$. Under a typing context Γ, process P has session type S. The typing rules are defined as follow:

$$\frac{}{\Gamma \vdash \mathbf{0} : end} \quad TY - End$$

$$\frac{\Gamma \vdash e : U \quad \Gamma \vdash P : S}{\Gamma \vdash \overline{\boldsymbol{p}}\langle e \rangle.P : \boldsymbol{p}![U];S} \quad TY - Send$$

$$\frac{\Gamma, x : U \vdash P : S}{\Gamma \vdash \boldsymbol{p}(x).P : \boldsymbol{p}?[U];S} \quad TY - Recv$$

$$\frac{\Gamma \vdash P : S}{\Gamma \vdash \boldsymbol{p} \triangleleft l.P : \boldsymbol{p} \oplus \{l : S\}} \quad TY - Sel$$

$$\frac{\forall i \in I.\ \Gamma \vdash P_i : S_i}{\Gamma \vdash \boldsymbol{p} \triangleright \{l_i : P_i\}_{i \in I} : \boldsymbol{p} \& \{l_i : S_i\}_{i \in I}} \quad TY - Bra$$

$$\frac{\Gamma \vdash P : S \quad S \leqslant S'}{\Gamma \vdash P : S'} \quad TY - Sub$$

$$\frac{\Gamma \vdash e : bool \quad \Gamma \vdash P : S \quad \Gamma \vdash Q : S}{\Gamma \vdash if\ e\ then\ P\ else\ Q : S} \quad TY - If$$

$$\frac{}{\Gamma, X : S \vdash X : S} \quad TY - PVar$$

$$\frac{\Gamma, X : S \vdash P : S}{\Gamma \vdash \mu X.P : S} \quad TY - Rec$$

In $TY - Recv$, the variable is removed from the typing context once the bound variable is used

$$\frac{\overline{date : string \vdash \mathbf{0} : end}}{\cdot \vdash \boldsymbol{Bob}(date).\mathbf{0} : \boldsymbol{Bob}?[string];end}$$

$TY - Sub$ states that a process P has session type S' if P has type S and S is a subtype of S'. This is usually used with $TY - Sel$ to add more choices to the result type $\boldsymbol{p} \oplus \{l : S\}$

Process P and process$Q$ needs to have the same session type S when using $TY - If$. The types can be made the same using, $TY - Sub$. If the session types cannot be convert to the same type using subtying, then it is not possible to assign a type to the process, e.g. we cannot assign a type to

$$if\ true\ then\ \overline{\boldsymbol{Bob}}\langle 3\rangle\ else\ \overline{\boldsymbol{Bob}}\langle "string"\rangle$$

1.8 Composing Processes

We say two processes are well typed if

$$\frac{\cdot \vdash P : S \quad \cdot \vdash Q : \overline{S}}{\vdash \boldsymbol{Alice} :: P \mid \boldsymbol{Bob} :: Q}$$

Meaning the two processes are dual type of each other.

The **Preservation Theorem** states that well-typeness is preserved during reduction, which is if $\mathcal{M}$ is well-typed, and $\mathcal{M} \to \mathcal{M}'$, then $\mathcal{M}'$ is also well-typed.

The **Progress Theorem** states that well-typed binary session does not gets stuck, it either reaches the end, or it can be further reduced. Formally, if $\mathcal{M}$ is well-typed, then there is a $\mathcal{M}'$ such that $\mathcal{M} \to \mathcal{M}'$ or $\mathcal{M} \equiv \boldsymbol{Alice} :: \mathbf{0} \mid \boldsymbol{Bob} :: \mathbf{0}$

2 Multiparty Session Types

In the previous chapter, we only looked at binary session types with two participants $\boldsymbol{Bob}$ and $\boldsymbol{Alice}$. This chapter extends the calculus to multiparty.

We first re-define the syntax of session to include more than two participants

$\mathcal{M}, \mathcal{M}' ::= \boldsymbol{p} :: P$	Single Process
$\mid \mathcal{M} \mid \mathcal{M}'$	Parallel composition

The following multiparty session syntax are structurally congruent

$$\mathcal{M} \mid \mathcal{M}' \equiv \mathcal{M}' \mid \mathcal{M}$$
$$\mathcal{M}_1 \mid (\mathcal{M}_2 \mid \mathcal{M}_3) \equiv (\mathcal{M}_1 \mid \mathcal{M}_2) \mid \mathcal{M}_3$$

$$\boldsymbol{p} :: \mathbf{0} \mid \mathcal{M} \equiv \mathcal{M}$$
$$P \equiv P' \to \boldsymbol{p} :: P \mid \mathcal{M} \equiv \boldsymbol{p} :: P' \mid \mathcal{M}$$

In Multiparty session type, we use a **Global Type** to describe the communications between the participants. We can then find the **Local Type** of each participant by projecting the Global type to the participant. Using the Local Type, each participant can implement their own **Local Process** independent of each other.

2.1 Global Type

The Global Type provides a bird's eye view of all the communication between participants. It is defined as

$G ::= end$	Termination
$\mid \boldsymbol{p} \to \boldsymbol{q} : [U]; G$	Message
$\mid \boldsymbol{p} \to \boldsymbol{q}\{l_i : G_i\}_{i \in I}$	Branching
$\mid \mu \boldsymbol{t}.\ G$	Recursive Type
$\mid \boldsymbol{t}$	Type Variable

$\boldsymbol{Alice} \to \boldsymbol{Bob} : [int]$ means $\boldsymbol{Alice}$ sends an integer to $\boldsymbol{Bob}$.

$\boldsymbol{Alice} \to \boldsymbol{Bob}\begin{cases} accept : \ ... \\ reject : \ ... \end{cases}$ means $\boldsymbol{Alice}$ selects one the options offered by $\boldsymbol{Bob}$.

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

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

3 Projection

We can find the Local Type of each participant by projecting the Global Type onto the participant. We write $G \mid \boldsymbol{p}$ as the projection of G to participant $\boldsymbol{p}$.

$$\boldsymbol{p} \to \boldsymbol{q} : [U]; G \mid \boldsymbol{r} = \begin{cases} \boldsymbol{q}![U]; (G \mid \boldsymbol{r}),\ \boldsymbol{r} = \boldsymbol{p} \\ \boldsymbol{p}?[U]; (G \mid \boldsymbol{r},\ \boldsymbol{r} = \boldsymbol{q} \\ G \mid \boldsymbol{r},\ \text{Otherwise} \end{cases}$$

From the perspective of $\boldsymbol{p}$, it is sending a message of sort U to $\boldsymbol{q}$.

From the perspective of $\boldsymbol{q}$, it is receiving a message of sort U from $\boldsymbol{p}$.

From another participant perspective, this interaction is unrelated.

$$\boldsymbol{p} \to \boldsymbol{q}\{l_i : G_i\}_{i\in I} \mid \boldsymbol{r} = \begin{cases} \boldsymbol{q} \oplus \{l_i : G_i \mid \boldsymbol{r}\}_{i\in I}, \ \boldsymbol{r} = \boldsymbol{p} \\ \boldsymbol{p} \& \{l_i : G_i \mid \boldsymbol{r}\}_{i\in I}, \ \boldsymbol{r} = \boldsymbol{q} \\ G_i \mid \boldsymbol{r}, \ \boldsymbol{r} \neq \boldsymbol{p}, \ \boldsymbol{r} \neq \boldsymbol{q}, \ \forall i, j \in I.\ G_i \mid \boldsymbol{r} = G_j \mid \boldsymbol{r} \\ undefined, \text{ Otherwise} \end{cases}$$

From the perspective of $\boldsymbol{p}$, it is making a choice offered by $\boldsymbol{q}$.

From the perspective of $\boldsymbol{q}$, it is offering selecting to $\boldsymbol{p}$.

From the perspective of another participant, if they are not involved in any of the branch then the interaction is unrelated. However, if the participant is involved in one of the branch, then the type is a **plain merge** of every branch.

For example, the global type

$$\boldsymbol{p} \to \boldsymbol{q} \begin{cases} yes : \boldsymbol{p} \to \boldsymbol{r} : [int]; end \\ no : \boldsymbol{p} \to \boldsymbol{r} : [int]; end \end{cases}$$

projected to $\boldsymbol{r}$ is $\boldsymbol{p}?[int]$ because $\boldsymbol{r}$ receives an integer from $\boldsymbol{p}$ in every branch.

If there are no common types involving $\boldsymbol{r}$ in every branch, then the projection is undefined. For example, the following global type projected to $\boldsymbol{r}$ is undefined.

$$\boldsymbol{p} \to \boldsymbol{q} \begin{cases} yes : \boldsymbol{p} \to \boldsymbol{r} : [string]; end \\ no : \boldsymbol{p} \to \boldsymbol{r} : [int]; end \end{cases} \qquad \boldsymbol{p} \to \boldsymbol{q} \begin{cases} yes : \boldsymbol{p} \to \boldsymbol{r} : [int]; end \\ no : end \end{cases}$$

3.1 Composing Processes

For multiparty session types, we use the judgement $\vdash \mathcal{M} : G$, which is derived from the typing rule

$$\frac{\forall i \in I \quad \cdot \vdash P_i : G \mid \boldsymbol{p}_i \quad pt(G) \subseteq \{\boldsymbol{p}_i \mid i \in I\}}{\vdash \prod_{i\in I} \boldsymbol{p}_i :: P_i : G}$$