Session Types Revisited

Ornela Dardha¹
School of Computing Science, University of Glasgow

(Based on joint work with Elena Giachino and Davide Sangiorgi)

Seminar at University of Nice / INRIA Sophia-Antipolis
16/11/2017

¹Supported by the UK EPSRC grant From Data Types to Session Types: A Basis for Concurrency and Distribution (EP/K034413/1)
Session Types in One Slide

• In distributed systems communicating peers agree on a protocol to follow, by specifying type and direction of data exchanged.

• Session types are a type formalism used to model structured communication-based programming.

• Guarantee privacy, communication safety and session fidelity.

• Designed for
  • $\pi$-calculus
  • functional languages
  • object-oriented languages
  • binary or multiparty communication
  • ... and so on!
Session Types

- Since their appearance, session types have developed into a significant theme in programming languages.
- Computing has moved from the era of data processing to the era of communication.
- Data types codify the structure of data and make it available to programming tools.
- Session types codify the structure of communication and make it available to programming tools.
The Maths Server and Client: Types / Protocols

Legend

- \&: branch/offfer/external choice;
- \⊕: select/internal choice;
- \?\text{Int}.\ T: input Int, continue as \(T\);
- !\text{Int}.\ T: output Int, continue as \(T\);
- “·” indicates sequencing;
- \textit{add}, \textit{neg}, \textit{quit}: choice labels, all different;
- \texttt{end} marks the end of the protocol.
The Maths Server and Client: Types / Protocols

- The session type $S$ of the server’s channel endpoint:

$$S \triangleq \{ \text{plus} : \text{?Int.} \rightarrow \text{Int.} \rightarrow \text{!Int.} \rightarrow \text{end},$$
$$\text{equal} : \text{?Int.} \rightarrow \text{Int.} \rightarrow \text{!Bool.} \rightarrow \text{end},$$
$$\text{neg} : \text{?Int.} \rightarrow \text{!Int.} \rightarrow \text{end} \}$$
The Maths Server and Client: Types / Protocols

- The session type $S$ of the server's channel endpoint:

  \[ S \triangleq \{ \begin{array}{l} \text{plus} : ?\text{Int}.?\text{Int}!\text{Int}.\text{end}, \\
                                 \text{equal} : ?\text{Int}.?\text{Int}!\text{Bool}.\text{end}, \\
                                 \text{neg} : ?\text{Int}!\text{Int}.\text{end} \end{array} \} \]

- The session type $C$ of the client's channel endpoint:

  \[ C \triangleq \{ \begin{array}{l} \text{plus} : !\text{Int}!\text{Int}.?\text{Int}.\text{end}, \\
                                 \text{equal} : !\text{Int}!\text{Int}.?\text{Bool}.\text{end}, \\
                                 \text{neg} : !\text{Int}.?\text{Int}.\text{end} \end{array} \} \]
The Maths Server and Client: Types / Protocols

• The session type $S$ of the server’s channel endpoint:

$$S \triangleq \&\{ \text{plus} : ?\text{Int.}!\text{Int.}.\text{end},$$
$$\quad \text{equal} : ?\text{Int.}!\text{Int.}.!\text{Bool.}.\text{end},$$
$$\quad \text{neg} : ?\text{Int.}!\text{Int.}.\text{end} \}$$

• The session type $C$ of the client’s channel endpoint:

$$C \triangleq \oplus\{ \text{plus} : !\text{Int.}!\text{Int.}!\text{Int.}.\text{end},$$
$$\quad \text{equal} : !\text{Int.}!\text{Int.}!\text{Int.}!\text{Bool.}.\text{end},$$
$$\quad \text{neg} : !\text{Int.}!\text{Int.}.\text{end} \}$$

Type duality: $S = \overline{C}$
The Maths Server: Process and Type

The server process operates on endpoint $x$ and is defined as follows:

\[
\text{server} \triangleq x \triangleright \{ \begin{array}{l}
\text{plus} : x?(v_1).x?(v_2).x!\langle v_1 + v_2 \rangle.0, \\
\text{equal} : x?(v_1).x?(v_2).x!\langle v_1 == v_2 \rangle.0, \\
\text{neg} : x?(v).x!\langle -v \rangle.0
\end{array} \}
\]

Session endpoint $x$ obeys:

\[
S \triangleq \& \{ \begin{array}{l}
\text{plus} : ?\text{Int.}?\text{Int.}!\text{Int.}\text{end}, \\
\text{equal} : ?\text{Int.}?\text{Int.}!\text{Bool.}\text{end}, \\
\text{neg} : ?\text{Int.}!\text{Int.}\text{end}
\end{array} \}
\]
The Maths Client: Process and Type

A client process operates on endpoint $y$ and can be defined as follows:

$$
client \triangleq y \triangleleft equal.y!\langle 3\rangle.y!\langle 5\rangle.y?(eq).0
$$

Session endpoint $y$ obeys:

$$
C \triangleq \oplus\{ plus : !\text{Int}.!\text{Int}.?\text{Int}.\text{end},
\quad equal : !\text{Int}.!\text{Int}.?\text{Bool}.\text{end},
\quad neg : !\text{Int}.?\text{Int}.\text{end} \}
$$
Client/Server Interaction
($\pi$-calculus OS)

($\nu xy : C)(server \mid client)$
Client/Server Interaction

($\pi$-calculus OS)

\[
(\nu xy : C)(\text{server} \mid \text{client})
\downarrow
(\nu xy : !\text{Int}!\text{Int}?\text{Bool}.\text{end})(x?(v_1).x?(v_2).x!\langle v_1 == v_2\rangle.0 \mid y!\langle 3\rangle.y!\langle 5\rangle.y?(eq).0)
\]
Client/Server Interaction
\((\pi\text{-calculus OS})\)

\[
(\nu xy : C)(server \mid client)
\]

\[
\downarrow
\]

\[
(\nu xy : !\text{Int}.!\text{Int}.?\text{Bool}.\text{end})(x?(v_1).x?(v_2).x!{v_1 == v_2}.0 \mid y!(3).y!(5).y?(eq).0)
\]

\[
\downarrow
\]

\[
(\nu xy : !\text{Int}.?\text{Bool}.\text{end})(x?(v_2).x!{3 == v_2}.0 \mid y!(5).y?(eq).0)
\]

\[
\equiv 0
\]
Client/Server Interaction
($\pi$-calculus OS)

\[
(\nu xy : C)(server \mid client) \\
\downarrow \\
(\nu xy : !\text{Int}!.\text{Int}.?\text{Bool}.\text{end})(x?(v_1).x?(v_2).x!(v_1 == v_2).0 \mid y!(3).y!(5).y?(eq).0) \\
\downarrow \\
(\nu xy : !\text{Int}!.?\text{Bool}.\text{end})(x?(v_2).x!(3 == v_2).0 \mid y!(5).y?(eq).0) \\
\downarrow \\
(\nu xy : ?\text{Bool}.\text{end})(x!(3 == 5).0 \mid y?(eq).0)
\]
Client/Server Interaction
($\pi$-calculus OS)

\[(\nu x y : C)(server \mid client)\]
\[\downarrow\]
\[(\nu x y : !\text{Int}.!\text{Int}.?\text{Bool}.\text{end})(x ?(v_1).x ?(v_2).x !\langle v_1 == v_2 \rangle.0 \mid y !\langle 3 \rangle.y !\langle 5 \rangle.y ?(eq).0)\]
\[\downarrow\]
\[(\nu x y : !\text{Int}.?\text{Bool}.\text{end})(x ?(v_2).x !\langle 3 == v_2 \rangle.0 \mid y !\langle 5 \rangle.y ?(eq).0)\]
\[\downarrow\]
\[(\nu x y : ?\text{Bool}.\text{end})(x !\langle 3 == 5 \rangle.0 \mid y ?(eq).0)\]
\[\downarrow\]
\[(\nu x y : \text{end})(0 \mid 0)\]
Client/Server Interaction
($\pi$-calculus OS)

\[
(\nu xy : C)(\text{server | client})
\]

\[
(\nu xy : !\text{Int}!\text{Int}?.\text{Bool}.\text{end})(x?(v_1).x?(v_2).x!\langle v_1 == v_2 \rangle.0 | y!\langle 3 \rangle.y!\langle 5 \rangle.y?(eq).0)
\]

\[
(\nu xy : !\text{Int}?.\text{Bool}.\text{end})(x?(v_2).x!\langle 3 == v_2 \rangle.0 | y!\langle 5 \rangle.y?(eq).0)
\]

\[
(\nu xy : ?\text{Bool}.\text{end})(x!\langle 3 == 5 \rangle.0 | y?(eq).0)
\]

\[
(\nu xy : \text{end})(0 | 0)
\]

\[\equiv 0\]
Session Types: Features

- **Duality**: the relationship between the types of opposite endpoints of a session channel.
- **Linearity**: each channel endpoint occurs exactly once in a collection of parallel processes.
- The **structure** of session types matches the structure of communication.
- Session types **evolve** as communication occurs.
Session Types: Properties

- Communication Safety: expected types of exchanged data.
- Session Fidelity: expected structure of session channels.
- Privacy: session channel owned *only* by communicating peers.
Session Types: Properties

- **Communication Safety**: expected types of exchanged data.
- **Session Fidelity**: expected structure of session channels.
- **Privacy**: session channel owned *only* by communicating peers.

**Main Theorem**: at runtime, communication follows the protocol.
A bit more formal...
The Session $\pi$-Calculus: Types

$S ::= \quad \text{end} \quad \text{termination}$  
$\quad \! T.S \quad \text{send}$  
$\quad ? T.S \quad \text{receive}$  
$\quad \oplus \{ l_i : S_i \}_{i \in I} \quad \text{select}$  
$\quad \& \{ l_i : S_i \}_{i \in I} \quad \text{branch}$

$T ::= \quad S \quad \text{session type}$  
$\quad \# T \quad \text{standard channel type}$  
$\quad \text{Bool} \quad \text{boolean type}$  
$\quad \ldots \quad \text{other type constructs}$
The Session $\pi$-Calculus: Terms

$P, Q ::= \begin{array}{ll}
0 & \text{inaction} \\
P \mid Q & \text{composition} \\
x!(v).P & \text{output} \\
x?(y).P & \text{input} \\
x \triangle l_j.P & \text{selection} \\
x \triangleright \{l_i : P_i\}_{i \in I} & \text{branching} \\
(\nu xy)P & \text{restriction}
\end{array}$

$\nu ::= x, y \quad \text{channel}$

true $|$ false $\quad$ boolean values

... $\quad$ other values
Session Typing Rules

(T-Res)
\[ \Gamma, x : S, y : \overline{S} \vdash P \]
\[ \Gamma \vdash (\nu xy)P \]

(T-In)
\[ \Gamma, x : S, y : T \vdash P \]
\[ \Gamma, x : ?T.S \vdash x?(y).P \]

(T-Out)
\[ \Gamma_1, x : S \vdash P \quad \Gamma_2 \vdash v : T \]
\[ (\Gamma_1, x : !T.S) + \Gamma_2 \vdash x!(v).P \]
Standard Typing Rules

(T-StndRes)
\[
\Gamma, x : T \vdash P \quad \text{\(T\) is not a session type}
\]
\[
\Gamma \vdash (\nu x)P
\]

(T-StndIn)
\[
\Gamma, x : \#T, y : T \vdash P
\]
\[
\Gamma, x : \#T \vdash x?(y).P
\]

(T-StndOut)
\[
\Gamma_1, x : \#T \vdash P \quad \Gamma_2 \vdash v : T
\]
\[
(\Gamma_1, x : \#T) + \Gamma_2 \vdash x!\langle v \rangle . P
\]
From Session Types to Standard \( \pi \)-Calculus Types
On standard types for \( \pi \)-calculus

- \( \#T \): channel used in input/output to transmit data of type \( T \).
  (Milner, Technical Report, LFCS’91)

- \( iT/oT \): channel used only in input/output to transmit data of type \( T \).
  (Pierce and Sangiorgi, LICS’93)

- \( ℓiT/ℓoT \): channel used only in input/output and exactly once to transmit data of type \( T \).
  (Kobayashi, Pierce and Turner, POPL’96)

- \( ⟨l_i:T_i⟩ \in I \): variant types: labelled disjoint union of types.
  (Sangiorgi, Info. & Comp.’98)
On standard types for $\pi$-calculus

- $\# T$: channel used in input/output to transmit data of type $T$. (Milner, Technical Report, LFCS'91)

- $iT/oT$: channel used only in input/output to transmit data of type $T$. (Pierce and Sangiorgi, LICS'93)

- $\ell_i T/\ell_o T$: channel used only in input/output and exactly once to transmit data of type $T$. (Kobayashi, Pierce and Turner, POPL'96)

- $\langle l_i : T \rangle_{i \in I}$: variant types: labelled disjoint union of types. (Sangiorgi, Info. & Comp.'98)
On standard types for $\pi$-calculus

- $\#T$: channel used in input/output to transmit data of type $T$. (Milner, Technical Report, LFCS'91)

- $iT/oT$: channel used only in input/output to transmit data of type $T$. (Pierce and Sangiorgi, LICS'93)

- $\ell_iT/\ell_oT$: channel used only in input/output and exactly once to transmit data of type $T$. (Kobayashi, Pierce and Turner, POPL'96)

- $\langle l_i : T \rangle_i \in I$: variant types: labelled disjoint union of types. (Sangiorgi, Info. & Comp.'98)
On standard types for $\pi$-calculus

- $\# T$: channel used in input/output to transmit data of type $T$. (Milner, Technical Report, LFCS'91)

- $iT/oT$: channel used only in input/output to transmit data of type $T$. (Pierce and Sangiorgi, LICS'93)

- $\ell_i T / \ell_o T$: channel used only in input/output and exactly once to transmit data of type $T$. (Kobayashi, Pierce and Turner, POPL'96)

- $\langle l_i : T_i \rangle_{i \in I}$ variant types: labelled disjoint union of types. (Sangiorgi, Info. & Comp.'98)
Session types vs. standard $\pi$-types

Keywords for session types:

1. Linearity
2. Duality
3. Restriction
4. Branch/Select

Keywords for standard $\pi$-types:

- Linearity forces a channel to be used exactly once.
- Capability of input/output of the same channel split between two peers.
- Restriction construct permits the creation of fresh private channels.
- Variant type permits choice.
Session types vs. standard $\pi$-types

Keywords for session types:

1. Linearity
2. Duality
3. Restriction
4. Branch/Select

Keywords for standard $\pi$-types:

1. **Linearity** forces a channel to be used exactly once.
2. **Capability** of input/output of the same channel split between two peers.
3. **Restriction** construct permits the creation of fresh private channels.
4. **Variant type** permits choice.
The Standard $\pi$-Calculus: Types

$$\tau ::= \begin{align*}
\ell_i[\tau] & \quad \text{linear input} \\
\ell_o[\tau] & \quad \text{linear output} \\
\ell_#[\tau] & \quad \text{linear connection} \\
\emptyset[] & \quad \text{channel with no capability} \\
\langle l_i : \tau_i \rangle_{i \in I} & \quad \text{variant type} \\
\#\tau & \quad \text{standard channel type} \\
\text{Bool} & \quad \text{boolean type} \\
\cdots & \quad \text{other constructs}
\end{align*}$$
The Standard $\pi$-Calculus: Terms

\[ P, Q ::= \begin{array}{c}
0 & \text{inaction} \\
x!\langle \tilde{v} \rangle . P & \text{output} \\
x?(\tilde{y}) . P & \text{input} \\
P | Q & \text{composition} \\
(\nu x)P & \text{channel restriction} \\
\text{case } v \text{ of } \{ l_i \cdot x_i \triangleright P_i \}_{i \in I} & \text{case process} \\
\end{array} \]

\[ v ::= x & \text{variable} \\
b & \text{boolean values} \\
l \cdot v & \text{variant value} \]
Bridging the gap between two worlds

Research Question: *To which extent session constructs are more complex and more expressive than the standard π-calculus constructs?*
Research Timeline

Milner, Parrow, Walker 1989

Honda 1993
- Takeuchi, Honda, Kubo 1994
  - Honda, Vasconcelos, Kubo 1998
    - To be continued...

Milner 1991
- Pierce, Sangiorgi 1993
  - Kobayashi, Pierce, Turner 1996
    - Sangiorgi 1998
      - Kobayashi 2007
        - Gay, Gesbert, Ravara 2008
          - Demangeon, Honda 2011
            - Dardha, Giachino, Sangiorgi 2012
              - Dardha 2014
Session Types Revisited

Session types are encoded into standard $\pi$-calculus types.

Encoding is based on:

1. **Linearity** of $\pi$-calculus channels.
2. **Input/Output** channel capabilities.
3. **Variant** types for the $\pi$-calculus.
4. **Continuation-Passing** principle.
Intuition of the Encoding

1. Session types encoded as linear $\pi$-calculus channel types.
2. ? and ! encoded as $\ell_i$ and $\ell_o$.
3. $\&\{l_i : S_i\}_{i \in I}$ and $\oplus\{l_i : S_i\}_{i \in I}$ encoded using variant types.
4. Continuation of a session type becomes payload type.
5. Dual actions in session continuation become equal payloads.
Why is this interesting?

Benefits of the encoding:

1. Bring session types to programming languages.
2. Bring standard typed $\pi$-calculus theory to session types.
3. Derivation of properties for session $\pi$-calculus from the standard typed $\pi$-calculus. (e.g. SR, TS)
4. Elimination of redundancy in the syntax of types and terms and in the theory.
5. Encoding is robust (subtyping, polymorphism, higher-order communication, recursion).
6. Expressivity result for session types.
Encoding Session Types: Example

Let

\[ S_{equal} = \text{?Int.} \text{?Int.} \text{!Bool.} \text{end} \]

Then

\[ [S_{equal}] = \ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]]] \]
Encoding Session Types: Example

Let

\[ S_{equal} = \textsf{?Int.}?\textsf{Int.}!\textsf{Bool.}\text{end} \]

Then

\[
[S_{equal}] = \ell_i[\textsf{Int}, \ell_i[\textsf{Int}, \ell_o[\textsf{Bool}, \emptyset][]]]
\]
Encoding Session Types: Example

Let

\[ S_{\text{equal}} = \text{?Int.}?\text{Int.}!\text{Bool}.\text{end} \]

Then

\[ [S_{\text{equal}}] = ℓ_i[\text{Int}, ℓ_i[\text{Int}, ℓ_o[\text{Bool}, \emptyset[]]]] \]
Let $S_{\text{equal}} = \text{?Int.?Int.}!\text{Bool}.\text{end}$

Then $\llbracket S_{\text{equal}} \rrbracket = \ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]]$
Encoding Session Types: Example

Let

\[ S_{equal} = ?\text{Int}.?\text{Int}.!\text{Bool} \text{end} \]

Then

\[ [S_{equal}] = \ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]] \]
Let
\[ C_{equal} = !\text{Int} \cdot !\text{Int} \cdot ?\text{Bool} \cdot \text{end} \]

Then
\[ \llbracket C_{equal} \rrbracket = \ell_o[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]] \]
What happens to duality?

The encoding of dual types is as follows:

\[
[S_{equal}] = \ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]] 
\]

and

\[
[C_{equal}] = \ell_o[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]] 
\]

Remark: duality on session types boils down to opposite capabilities (\(\ell_i/\ell_o\)) only in the outermost level!
What happens to duality?

The encoding of dual types is as follows:

\[
[S_{equal}] = \ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]]
\]

and

\[
[C_{equal}] = \ell_o[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]]
\]

Remark

duality on session types boils down to opposite capabilities (\(\ell_i/\ell_o\)) of channel types, **only** in the outermost level!
Encoding the Maths Server and Client: Types

\[ S \triangleq \{ \text{plus} : \text{?Int.}\text{?Int.}\text{!!Int.}\text{end}, \]
\[ \text{equal} : \text{?Int.}\text{?Int.}\text{!!Bool.}\text{end}, \]
\[ \text{neg} : \text{?Int.}\text{!!Int.}\text{end} \} \]

\([S] = \ell_i[\langle \text{plus}_\ell ; \ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Int}, \emptyset[\text{}]]]]
[\text{equal}_\ell ; \ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[\text{}]]]]],
[\text{neg}_\ell ; \ell_i[\text{Int}, \ell_o[\text{Int}, \emptyset[\text{}]]] \rangle] \]
Encoding the Maths Server and Client:

Types

\[ C \triangleq \oplus \{ \]

\begin{align*}
  plus & : !\text{Int}!\text{Int}?\text{Int}.\text{end}, \\
  equal & : !\text{Int}!\text{Int}?\text{Bool}.\text{end}, \\
  neg & : !\text{Int}?\text{Int}.\text{end} \}
\end{align*}

\[
\llbracket C \rrbracket = \ell_o[\langle \]

\begin{align*}
  plus_\ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Int}, \emptyset[]]]], \\
  equal_\ell_i[\text{Int}, \ell_i[\text{Int}, \ell_o[\text{Bool}, \emptyset[]]]], \\
  neg_\ell_i[\text{Int}, \ell_o[\text{Int}, \emptyset[]]] \rangle]
\end{align*}

\]
 Encoding of Session Types: Formally

\[
\begin{align*}
\text{[end]} & \triangleq \emptyset[\cdot] \\
[! T . S] & \triangleq \ell_o[[T], [\overline{S}]] \\
[? T . S] & \triangleq \ell_i[[T], [S]] \\
\bigoplus\{l_i : S_i\}_{i \in I} & \triangleq \ell_o[\langle l_i : [\overline{S_i}] \rangle_{i \in I}] \\
\&\{l_i : S_i\}_{i \in I} & \triangleq \ell_i[\langle l_i : [S_i] \rangle_{i \in I}]
\end{align*}
\]
Renaming function is the identity function $\emptyset$:

$$\llbracket (\nu xy)(server \mid client) \rrbracket_{\emptyset} = (\nu z)\llbracket (server \mid client) \rrbracket_{\{x, y \mapsto z\}}$$

$$= (\nu z)(\llbracket server \rrbracket_{\{x \mapsto z\}} \mid \llbracket client \rrbracket_{\{y \mapsto z\}})$$

$\{x, y \mapsto z\}$ maps $x$ and $y$ to a fresh name $z$;
Encoding the Maths Server and Client: Processes

\[
\begin{align*}
\llbracket \text{server} \rrbracket \{ x \mapsto z \} &= \\
z?(y).\text{case } y \text{ of } \{ \\
\quad \text{plus\_}(s) &\triangleright s?(v_1, c).c?(v_2, c').(\nu c'')c'!\langle v_1 + v_2, c'' \rangle.0 \\
\quad \text{equal\_}(s) &\triangleright s?(v_1, c).c?(v_2, c').(\nu c'')c'!\langle v_1 == v_2, c'' \rangle.0 \\
\quad \text{neg\_}(s) &\triangleright s?(v, c).(\nu c'')c!\langle -v, c'' \rangle.0 \\
\} \end{align*}
\]

\[
\begin{align*}
\llbracket \text{client} \rrbracket \{ y \mapsto z \} &= \\
(\nu s)z!\langle \text{equal\_} s \rangle.(\nu c)s!\langle 3, c \rangle.(\nu c')c!\langle 5, c' \rangle.c'?\langle eq, c'' \rangle.0
\end{align*}
\]
Encoding of Session Terms: Formally

\[ [x! \langle \nu \rangle . P]_f \triangleq (\nu c) f_x ! \langle [\nu]_f, c \rangle . [P]_f, \{ x \mapsto c \} \]

\[ [x?(y). P]_f \triangleq f_x ? (y, c) . [P]_f, \{ x \mapsto c \} \]

\[ [x \triangle l_j . P]_f \triangleq (\nu c) f_x ! \langle l_j - c \rangle . [P]_f, \{ x \mapsto c \} \]

\[ [x \triangleright \{ l_i : P_i \}_{i \in I}]_f \triangleq f_x ? (y). \text{ case } y \text{ of } \{ l_i - c \triangleright [P_i]_f, \{ x \mapsto c \} \}_{i \in I} \]

\[ [P \mid Q]_f \triangleq [P]_f \mid [Q]_f \]

\[ [\nu xy] P]_f \triangleq (\nu z) [P]_f, \{ x, y \mapsto z \} \]
Theorem

*Encoding preserves typability of programs.*

Theorem

*Encoding preserves evaluation of programs.*

Lemma

*Encoding of dual session types gives dual linear \( \pi \)-types.*
Session Types Revisited: Extensions and Implementations
Extensions of the Encoding of Session Types

Derived from the encoding:

- Subtyping of session channel types (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- Parametric and bounded polymorphism (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- Higher-order communication (Dardha, BEAT’14)

- Recursion (Dardha, BEAT’14)

- Progress, deadlock and lock freedom (Carbone, Dardha and Montesi, COORDINATION’14)
Extensions of the Encoding of Session Types

Derived from the encoding:

- **Subtyping of session channel types** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Parametric and bounded polymorphism** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)
Extensions of the Encoding of Session Types

Derived from the encoding:

- **Subtyping of session channel types** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Parametric and bounded polymorphism** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Higher-order communication** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)
Extensions of the Encoding of Session Types

Derived from the encoding:

- **Subtyping of session channel types** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Parametric and bounded polymorphism** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Higher-order communication** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Recursion** (Dardha, BEAT’14)
Extensions of the Encoding of Session Types

Derived from the encoding:

- **Subtyping of session channel types** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Parametric and bounded polymorphism** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Higher-order communication** (Dardha, Giachino and Sangiorgi, PPDP’12 / Info. & Comput.’17)

- **Recursion** (Dardha, BEAT’14)

- **Progress, deadlock and lock freedom** (Carbone, Dardha and Montesi, COORDINATION’14)
Mainstream Programming Languages with Session Types

**OCaml: FuSe**

- *A simple library implementation of binary sessions* (Padovani, Journal of Functional Programming’17)
- Lightweight OCaml module that implements binary session-based communications.
- Based on the continuation-passing principle and encoding (Kobayashi, Extended Report’07; Dardha, Giachino and Sangiorgi, PPDP’12)
- Static check of message ordering and dynamic linearity check.
Mainstream Programming Languages with Session Types

**Scala: lchannels**

- *Lightweight Session Programming in Scala* (Scalas and Yoshida, ECOOP’16)
- Lightweight binary session programming in Scala via lchannels library.
- Based on the continuation-passing principle and encoding (Kobayashi, Extended Report’07; Dardha, Giachino and Sangiorgi, PPDP’12)
- Static check of message ordering and dynamic linearity check.
Mainstream Programming Languages with Session Types

Scala

- *A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming* (Scalas, Dardha, Hu and Yoshida, ECOOP'17)

- **First** encoding of a *full-fledged* multiparty session types into linear $\pi$-calculus, following (Kobayashi, Extended Report’07; Dardha, Giachino and Sangiorgi, PPDP'12)

- Encoding preserves *distributivity* of multiparty session types.

- Automated generation of Scala APIs for multiparty sessions.

- The **first** artifact supporting *distributed multiparty delegation*: for free from the encoding!
Conclusions

- Encoding of session types into standard $\pi$-calculus types: compilation of session types into an “assembly” language.
- Encoding proved faithful: derive properties of session from the corresponding ones of standard $\pi$-calculus.
- Encoding proved robust for advanced features like subtyping, polymorphism, HO communication and recursion.
- Encoding is at the foundation for several implementations of session types in mainstream programming languages.
Audience! ⟨ThankYou⟩.
rec X {
    & {
        more_questions : Audience?(y : Question).Audience! ⟨Answer⟩.X, quit : end
    }
}
Extensions of the Encoding of Sessions
Subtyping

$S \leq T$

Safe Substitutability (Liskov and Wing 1994): "it is safe to use a value of type $S$ where a value of type $T$ is expected".

Meaning: No violation of the runtime safety that the type system guarantees.
Subtyping

\[ S \leq T \]

- **Safe Substitutability** (Liskov and Wing 1994): “it is safe to use a value of type \( S \) where a value of type \( T \) is expected”.

- ...Meaning: No violation of the runtime safety that the type system guarantees.
Updating the Maths Server by Subtyping

The old session type of the server’s endpoint:

\[ S_{\text{old}} \triangleq \{ \text{plus : } ?\text{Int.}\text{.Int.}!\text{Int.}\text{.end}, \text{equal : } ?\text{Int.}\text{.Int.}!\text{Bool.}\text{.end}, \text{neg : } ?\text{Int.}!\text{Int.}\text{.end} \} \]
Updating the Maths Server by Subtyping

The old session type of the server’s endpoint:

\[ S_{old} \triangleq \&\{ \quad plus : ?Int.?Int.!Int.end, \]
\[ \quad equal : ?Int.?Int.!Bool.end, \]
\[ \quad neg : ?Int.!Int.end \quad \} \]

The new session type of the server’s endpoint:

\[ S_{new} \triangleq \&\{ \quad mul : ?Int.?Int.!Int..end, \]
\[ \quad plus : ?Int.?Int.!Int.end, \]
\[ \quad equal : ?Int.?Int.!Bool.end, \]
\[ \quad neg : ?Int.!Int.end \quad \} \]

Subtyping for Session Types in the Pi Calculus. (Gay and Hole, Acta Informatica’05)
Subtyping by Encoding

Theorem

For all session types $S, S'$. $S <: S'$ if and only if $\llbracket S \rrbracket \leq \llbracket S' \rrbracket$. 
Subtyping by Encoding

Theorem

For all session types $S, S'$. $S <: S'$ if and only if $\llbracket S \rrbracket \leq \llbracket S' \rrbracket$.

Derived from the encoding and subtyping in standard typed $\pi$-calculus:

- Reflexivity and transitivity of subtyping for sessions.
- Lemmas (e.g., Substitution...) from the corresponding ones in the $\pi$-calculus: derived for free!
On Liveness Properties: Deadlock Freedom, Lock Freedom and Progress
Progress

- Progress is a fundamental property of safe processes.
Progress

- **Progress** is a fundamental property of safe processes.

- A program having progress does not get "stuck", i.e., a state that is not designated as a final value and that the language semantics does not tell how to evaluate further.
Comparing Properties of Communication

- **Deadlock Freedom**: communications eventually succeed, unless the whole process diverges. (*Standard $\pi$*)

- **Lock Freedom**: communications eventually succeed even if the whole process diverges. (*Standard $\pi$*)

- **Progress**: In-session communications eventually succeed, provided that a suitable context can be found. (*Session $\pi$*)
Deadlock Freedom vs. Lock Freedom

- Consider the process:

\[ P = (\nu xy)(\nu vw)(x? (z).v!\langle z \rangle \mid w?(u).y!\langle u \rangle) \]

It is deadlocked and hence locked!
\section*{Deadlock Freedom vs. Lock Freedom}

- Consider the process:

\[ P = (\nu xy)(\nu vw)(x?(z).\nu!\langle z \rangle \mid w?(u).y!\langle u \rangle) \]

It is deadlocked and hence locked!

- Consider the process:

\[ Q = (\nu xy)(x?(z) \mid \Omega) \]

It is deadlock-free but locked!
Research Question

What is the relationship among deadlock freedom, lock freedom and progress?
Research Question

What is the relationship among deadlock freedom, lock freedom and progress?

- Lock freedom is a stronger property than deadlock freedom.
Research Question

What is the relationship among deadlock freedom, lock freedom and progress?

• Lock freedom is a stronger property than deadlock freedom.

• Progress is a compositional form of lock freedom.
  (Carbone, Dardha and Montesi, COORDINATION’14)
New Type System for Progress

• Checking progress reduces to checking lock freedom for the catalyser closure of a process.

• Static analysis for lock-freedom lifted to static analysis for progress.

• E.g., we use Kobayashi’s typing discipline for lock-freedom in the standard typed $\pi$-calculus.

• We use our encoding of session $\pi$-calculus to the standard typed $\pi$-calculus.
1: procedure $\text{PROGRESS}(\Gamma, P)$
2: Check $\Gamma \vdash P$
3: Build close($P$) from $\Gamma$
4: Encode $[\text{close}(P)]_f = P'$
5: return $\text{TyPiCal}(P')$
6: end procedure


Ornela Dardha.  

Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida.  
A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.  
In *31st European Conference on Object-Oriented Programming, ECOOP*, Accepted for Publication.

Simon J. Gay and Malcolm Hole.  
Subtyping for session types in the pi calculus.  
Vasco T. Vasconcelos.
Fundamentals of session types.

Naoki Kobayashi.
A type system for lock-free processes.

Naoki Kobayashi.
A new type system for deadlock-free processes.

Naoki Kobayashi.
Type systems for concurrent programs.
References IV

Naoki Kobayashi. 
Type systems for concurrent programs. 
Extended version of [8], Tohoku University, 2007.

TYPICAL. 
Type-based static analyzer for the pi-calculus. 
http://www-kb.is.s.u-tokyo.ac.jp/~koba/typical/.

Luca Padovani. 
Fuse - a simple library implementation of binary sessions, 2016.

Alceste Scalas and Nobuko Yoshida. 
Lightweight session programming in scala. 