Introduction to Session Types

Ornela Dardha[1]

School of Computing Science
University of Glasgow, UK

BETTY Summer School 2016

[1] UK EPSRC project *From Data Types to Session Types: A Basis for Concurrency and Distribution* (EP/K034413/1)

Session Types in One Slide

- In complex distributed systems communicating participants agree on a protocol to follow, 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
  - π- calculus
  - functional languages
  - object-oriented languages
  - binary or multiparty communication
  - ...

Outline

Session Types[2]

- *Session types* were born more than 20 years ago.
- The π-calculus is the original and most used framework.
- The seminal works:
  - Honda, "*Types for Dyadic Interaction*", CONCUR 1993.
  - Takeuchi, Honda & Kubo, "*An Interaction-Based Language and its Typing System*", PARLE 1994.
  - Honda, Vasconcelos & Kubo, "*Language Primitives and Type Discipline for Structured Communication-Based Programming*", ESOP 1998.

[2] I thank Simon J. Gay for borrowing some of his slides

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.

Outline

The Maths Server and Client: Types / Protocols

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

$$
\begin{aligned}
S \quad \triangleq \quad \&\{ \; & \mathit{add} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.S, \\
& \mathit{neg} : ?\mathtt{Int}.!\mathtt{Int}.S \\
& \mathit{quit} : \mathtt{end} \; \}
\end{aligned}
$$

The Maths Server and Client: Types / Protocols

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

$$
\begin{aligned}
S \quad \triangleq \quad \&\{ \; & \mathit{add} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.S, \\
& \mathit{neg} : ?\mathtt{Int}.!\mathtt{Int}.S \\
& \mathit{quit} : \mathsf{end} \; \}
\end{aligned}
$$

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

$$
\begin{aligned}
C \quad \triangleq \quad \oplus\{ \; & \mathit{add} : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Int}.C, \\
& \mathit{neg} : !\mathtt{Int}.?\mathtt{Int}.C \\
& \mathit{quit} : \mathsf{end} \; \}
\end{aligned}
$$

The Maths Server and Client: Types / Protocols

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

$$
\begin{aligned}
S \quad \triangleq \quad \&\{ \; & add : \texttt{?Int.?Int.!Int}.S, \\
& neg : \texttt{?Int.!Int}.S \\
& quit : \texttt{end} \; \}
\end{aligned}
$$

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

$$
\begin{aligned}
C \quad \triangleq \quad \oplus\{ \; & add : \texttt{!Int.!Int.?Int}.C, \\
& neg : \texttt{!Int.?Int}.C \\
& quit : \texttt{end} \; \}
\end{aligned}
$$

Duality: $S = \overline{C}$

The Maths Server and Client: Types / Protocols

Legend

- &: branch/offer/external choice;
- $\oplus$: select/internal choice;
- ?`Int`.T: input `Int`, continue as T;
- !`Int`.T: output `Int`, continue as T;
- "." indicates sequencing;
- *add*, *neg*, *quit*: choice labels, all different;
- `end` marks the end of the protocol.

The Maths Server: Program and Type

A server *srv*, parametrised in its channel endpoint x of type S:

$$\begin{aligned} srv(x : S) \quad = x \triangleright \{\, & add : x?(a : \mathtt{Int}).x?(b : \mathtt{Int}).x!\langle a + b\rangle.srv(x), \\ & neg : x?(a : \mathtt{Int}).x!\langle -a\rangle.srv(x) \\ & quit : \mathbf{0} \ \} \end{aligned}$$

$$\begin{aligned} S \quad = \& \ \{ \ & add : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.S, \\ & neg : ?\mathtt{Int}.!\mathtt{Int}.S \\ & quit : \mathsf{end} \ \} \end{aligned}$$

The Maths Client: Program and Type

A client *clt*, parametrised in its channel endpoint x of type C:

$$clt(x : C) \quad = x \triangleleft neg.x!\langle 2\rangle.x?(a : \mathtt{Int}).x \triangleleft quit.P(a)$$

$$\begin{aligned} C \quad = \oplus\{\, & add : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Int}.C, \\ & neg : !\mathtt{Int}.?\mathtt{Int}.C \\ & quit : \mathtt{end} \,\} \end{aligned}$$

Client/Server Interaction
(π- calculus OS)

$$(\nu c : S)(srv(c^{+}) \mid clt(c^{-}))$$

Client/Server Interaction (π- calculus OS)

$$(\nu c : S)(srv(c^+) \mid clt(c^-))$$
$$\downarrow$$
$$(\nu c : ?\mathtt{Int}.!\mathtt{Int}.S)(c^+?(a : \mathtt{Int}).c^+!\langle -a\rangle.srv(c^+) \mid c^-!\langle 2\rangle.c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a))$$

Client/Server Interaction (π- calculus OS)

$$(\nu c : S)(srv(c^+) \mid clt(c^-))$$
$$\downarrow$$
$$(\nu c : ?\mathtt{Int}.!\mathtt{Int}.S)(c^+?(a : \mathtt{Int}).c^+!\langle -a\rangle.srv(c^+) \mid c^-!\langle 2\rangle.c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a))$$
$$\downarrow$$
$$(\nu c : !\mathtt{Int}.S)(c^+!\langle -2\rangle.srv(c^+) \mid c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a))$$

Client/Server Interaction (π- calculus OS)

$$
\begin{array}{c}
(\nu c : S)(srv(c^+) \mid clt(c^-)) \\
\downarrow \\
(\nu c : ?\mathtt{Int}.!\mathtt{Int}.S)(c^+?(a : \mathtt{Int}).c^+!\langle -a\rangle.srv(c^+) \mid c^-!\langle 2\rangle.c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a)) \\
\downarrow \\
(\nu c : !\mathtt{Int}.S)(c^+!\langle -2\rangle.srv(c^+) \mid c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a)) \\
\downarrow \\
(\nu c : S)(srv(c^+) \mid c^- \triangleleft quit.P(-2))
\end{array}
$$

Client/Server Interaction (π- calculus OS)

$$
\begin{gathered}
(\nu c : S)(srv(c^+) \mid clt(c^-)) \\
\downarrow \\
(\nu c : ?\mathtt{Int}.!\mathtt{Int}.S)(c^+?(a : \mathtt{Int}).c^+!\langle -a \rangle.srv(c^+) \mid c^-!\langle 2 \rangle.c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a)) \\
\downarrow \\
(\nu c : !\mathtt{Int}.S)(c^+!\langle -2 \rangle.srv(c^+) \mid c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a)) \\
\downarrow \\
(\nu c : S)(srv(c^+) \mid c^- \triangleleft quit.P(-2)) \\
\downarrow \\
(\nu c : \mathsf{end})(\mathbf{0} \mid P(-2))
\end{gathered}
$$

Client/Server Interaction (π- calculus OS)

$$
\begin{array}{c}
(\nu c : S)(srv(c^+) \mid clt(c^-)) \\
\downarrow \\
(\nu c : ?\mathtt{Int}.!\mathtt{Int}.S)(c^+?(a : \mathtt{Int}).c^+!\langle -a\rangle.srv(c^+) \mid c^-!\langle 2\rangle.c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a)) \\
\downarrow \\
(\nu c : !\mathtt{Int}.S)(c^+!\langle -2\rangle.srv(c^+) \mid c^-?(a : \mathtt{Int}).c^- \triangleleft quit.P(a)) \\
\downarrow \\
(\nu c : S)(srv(c^+) \mid c^- \triangleleft quit.P(-2)) \\
\downarrow \\
(\nu c : \mathrm{end})(\mathbf{0} \mid P(-2)) \\
\equiv \\
P(-2)
\end{array}
$$

Establishing a Connection

- The server listens on a standard channel a of type $\sharp S$, and receives a session channel for *srv* to use.

$$\mathit{server}(a) \quad = \quad a?(x : S).\mathit{srv}(x)$$

Establishing a Connection

- The server listens on a standard channel a of type $\sharp S$, and receives a session channel for srv to use.

$$server(a) \quad = \quad a?(x : S).srv(x)$$

- The global declaration $a : \sharp S$ advertises the server and its protocol.

Establishing a Connection

- The server listens on a standard channel a of type $\sharp S$, and receives a session channel for srv to use.

$$server(a) \quad = \quad a?(x : S).srv(x)$$

- The global declaration $a : \sharp S$ advertises the server and its protocol.
- The client creates a session channel and sends it to the server.

$$client(a) \quad = \quad (\nu c : S)(a!\langle c^+ \rangle.clt(c^-))$$

Establishing a Connection

- The server listens on a standard channel a of type $\sharp S$, and receives a session channel for srv to use.

$$server(a) \quad = \quad a?(x : S).srv(x)$$

- The global declaration $a : \sharp S$ advertises the server and its protocol.
- The client creates a session channel and sends it to the server.

$$client(a) \quad = \quad (\nu c : S)(a!\langle c^+ \rangle.clt(c^-))$$

- After one step, execution proceeds as before.

Outline

Session Types: Key Features

- Duality: the relationship between the types of opposite endpoints of a session channel.

Session Types: Key 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.

Session Types: Key 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: Key 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 change as communication occurs.

Session Types: Key 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 change as communication occurs.
- Connection is established among participants.

Properties of Session Types

- Communication Safety: the exchanged data has the expected type.

Properties of Session Types

- Communication Safety: the exchanged data has the expected type.
- Session Fidelity: the session channel has the expected structure.

Properties of Session Types

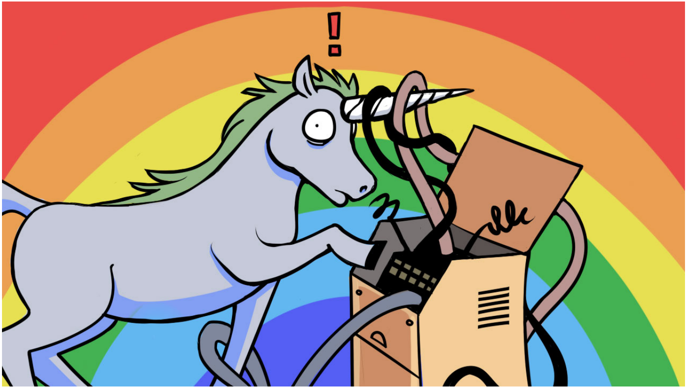
- Communication Safety: the exchanged data has the expected type.
- Session Fidelity: the session channel has the expected structure.
- Privacy: the session channel is owned only by the communicating parties.

Properties of Session Types

- Communication Safety: the exchanged data has the expected type.
- Session Fidelity: the session channel has the expected structure.
- Privacy: the session channel is owned only by the communicating parties.

Main Theorem: at runtime, communication follows the protocol.

The Calculus and Typing Rules



The Calculus: Types

S	$::=$	$\mathtt{end}$	termination
		$!T.S$	send
		$?T.S$	receive
		$\oplus\{l_i : S_i\}_{i \in I}$	select
		$\&\{l_i : S_i\}_{i \in I}$	branch
T	$::=$	S	session type
		$\mathtt{Bool}$	boolean type
		$\sharp T$	standard channel type
		$\ldots$	other type constructs

The Calculus: Terms

P, Q	$::=$	$\mathbf{0}$	inaction
		$P \mid Q$	composition
		$(\boldsymbol{\nu} x)P$	restriction
		$x^p!\langle v^q \rangle.P$	output
		$x^p?(y).P$	input
		$x^p \triangleleft l_j.P$	selection
		$x^p \triangleright \{l_i : P_i\}_{i \in I}$	branching
v	$::=$	x, y	channel
		$\texttt{true} \mid \texttt{false}$	boolean values
		$\ldots$	other values

p, q are optional *polarities* for *channels*, being $+$ or $-$

Typing Rules

$$\text{(T-PAR)}\quad \frac{\Gamma_1 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}$$

$$\text{(T-RES)}\quad \frac{\Gamma, x^+ : S, x^- : \overline{S} \vdash P}{\Gamma \vdash (\boldsymbol{\nu} x)P}$$

$$\text{(T-IN)}\quad \frac{\Gamma, x^p : S, y : T \vdash P}{\Gamma, x^p : ?T.S \vdash x^p?(y).P}$$

$$\text{(T-OUT)}\quad \frac{\Gamma_1, x^p : S \vdash P \quad \Gamma_2 \vdash v^q : T}{(\Gamma_1, x^p : !T.S) + \Gamma_2 \vdash x^p!\langle v^q \rangle.P}$$

$$\text{(T-BRCH)}\quad \frac{\Gamma, x^p : S_i \vdash P_i \quad \forall i \in I}{\Gamma, x^p : \&\{l_i : S_i\}_{i \in I} \vdash x^p \triangleright \{l_i : P_i\}_{i \in I}}$$

$$\text{(T-SEL)}\quad \frac{\Gamma, x^p : S_j \vdash P \quad j \in I}{\Gamma, x^p : \oplus\{l_i : S_i\}_{i \in I} \vdash x^p \triangleleft l_j.P}$$

Typing Rules

(T-Par)
$$\frac{\Gamma_1 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}$$

(T-Res)
$$\frac{\Gamma, x^+ : S, x^- : \overline{S} \vdash P}{\Gamma \vdash (\boldsymbol{\nu} x)P}$$

(T-In)
$$\frac{\Gamma, x^p : S, y : T \vdash P}{\Gamma, x^p : ?T.S \vdash x^p?(y).P}$$

(T-Out)
$$\frac{\Gamma_1, x^p : S \vdash P \quad \Gamma_2 \vdash v^q : T}{(\Gamma_1, x^p : !T.S) + \Gamma_2 \vdash x^p!\langle v^q \rangle.P}$$

(T-Brch)
$$\frac{\Gamma, x^p : S_i \vdash P_i \quad \forall i \in I}{\Gamma, x^p : \&\{l_i : S_i\}_{i \in I} \vdash x^p \triangleright \{l_i : P_i\}_{i \in I}}$$

(T-Sel)
$$\frac{\Gamma, x^p : S_j \vdash P \quad j \in I}{\Gamma, x^p : \oplus\{l_i : S_i\}_{i \in I} \vdash x^p \triangleleft l_j.P}$$

Gay & Hole, "*Subtyping for Session Types in the Pi Calculus*". ESOP 1999, Acta Informatica 2005.

Combination of Typing Contexts

$$\begin{aligned}
\Gamma + x^+ : S &= \Gamma, x^+ : S && \text{if } x, x^+ \notin \mathrm{dom}(\Gamma) \\
\Gamma + x^- : S &= \Gamma, x^- : S && \text{if } x, x^- \notin \mathrm{dom}(\Gamma) \\
\Gamma + x : T &= \Gamma, x : T && \text{if } x, x^+, x^- \notin \mathrm{dom}(\Gamma) \\
(\Gamma, x : T) + x : T &= \Gamma, x : T && \text{if T is not a session type}
\end{aligned}$$

Exercise: Is it well typed?

$$(\nu x)(x^{+}?(t : \texttt{Bool}).\mathbf{0} \mid x^{-}!\langle \texttt{true} \rangle.\mathbf{0})$$

Exercise: Is it well typed?

$$(\boldsymbol{\nu} x)(x^{+}?(t : \texttt{Bool}).\mathbf{0} \mid x^{-}!\langle \texttt{true} \rangle.\mathbf{0}) \qquad \checkmark$$

Exercise: Is it well typed?

$$(\nu x)(x^{+}?(t : \mathtt{Bool}).\mathbf{0} \mid x^{-}!\langle \mathtt{true} \rangle.\mathbf{0}) \qquad \checkmark$$

$$(\nu x)(x^{+}!\langle t \rangle.\mathbf{0} \mid x^{-}!\langle \mathtt{true} \rangle.\mathbf{0})$$

Exercise: Is it well typed?

$$(\boldsymbol{\nu} x)(x^{+}?(t : \texttt{Bool}).\mathbf{0} \mid x^{-}!\langle\texttt{true}\rangle.\mathbf{0}) \qquad \checkmark$$

$$(\boldsymbol{\nu} x)(x^{+}!\langle t\rangle.\mathbf{0} \mid x^{-}!\langle\texttt{true}\rangle.\mathbf{0}) \qquad \times$$

Exercise: Is it well typed?

$$(\boldsymbol{\nu} x)(x^{+}?(t : \texttt{Bool}).\mathbf{0} \mid x^{-}!\langle \texttt{true} \rangle.\mathbf{0})$$ ✓

$$(\boldsymbol{\nu} x)(x^{+}!\langle t \rangle.\mathbf{0} \mid x^{-}!\langle \texttt{true} \rangle.\mathbf{0})$$ ×

$$(\boldsymbol{\nu} x)(x^{-}!\langle \texttt{false} \rangle.\mathbf{0} \mid x^{+}?(t : \texttt{Bool}).\mathbf{0} \mid x^{+}?(w : \texttt{Bool}).\mathbf{0})$$

Exercise: Is it well typed?

$$(\boldsymbol{\nu} x)(x^{+}?(t : \mathtt{Bool}).\mathbf{0} \mid x^{-}!\langle \mathtt{true} \rangle.\mathbf{0}) \qquad \checkmark$$

$$(\boldsymbol{\nu} x)(x^{+}!\langle t \rangle.\mathbf{0} \mid x^{-}!\langle \mathtt{true} \rangle.\mathbf{0}) \qquad \times$$

$$(\boldsymbol{\nu} x)(x^{-}!\langle \mathtt{false} \rangle.\mathbf{0} \mid x^{+}?(t : \mathtt{Bool}).\mathbf{0} \mid x^{+}?(w : \mathtt{Bool}).\mathbf{0}) \qquad \times$$

Exercise: Is it well typed?

$$(\boldsymbol{\nu} x)(x^{+}?(t : \texttt{Bool}).\mathbf{0} \mid x^{-}!\langle \texttt{true} \rangle.\mathbf{0})$$ ✓

$$(\boldsymbol{\nu} x)(x^{+}!\langle t \rangle.\mathbf{0} \mid x^{-}!\langle \texttt{true} \rangle.\mathbf{0})$$ ×

$$(\boldsymbol{\nu} x)(x^{-}!\langle \texttt{false} \rangle.\mathbf{0} \mid x^{+}?(t : \texttt{Bool}).\mathbf{0} \mid x^{+}?(w : \texttt{Bool}).\mathbf{0})$$ ×

$$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)(y^{-}!\langle 42 \rangle.x^{+}?(z : \texttt{Int}).\mathbf{0} \mid x^{-}!\langle 11 \rangle.y^{+}?(w : \texttt{Int}).\mathbf{0})$$

Exercise: Is it well typed?

$$(\boldsymbol{\nu} x)(x^+?(t : \texttt{Bool}).\mathbf{0} \mid x^-!\langle\texttt{true}\rangle.\mathbf{0}) \qquad \checkmark$$

$$(\boldsymbol{\nu} x)(x^+!\langle t\rangle.\mathbf{0} \mid x^-!\langle\texttt{true}\rangle.\mathbf{0}) \qquad \times$$

$$(\boldsymbol{\nu} x)(x^-!\langle\texttt{false}\rangle.\mathbf{0} \mid x^+?(t : \texttt{Bool}).\mathbf{0} \mid x^+?(w : \texttt{Bool}).\mathbf{0}) \qquad \times$$

$$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)(y^-!\langle 42\rangle.x^+?(z : \texttt{Int}).\mathbf{0} \mid x^-!\langle 11\rangle.y^+?(w : \texttt{Int}).\mathbf{0}) \qquad \checkmark$$

Exercise: Is it well typed?

$$(\nu x)(x^+?(t : \texttt{Bool}).\mathbf{0} \mid x^-!\langle \texttt{true} \rangle.\mathbf{0}) \quad \checkmark$$

$$(\nu x)(x^+!\langle t \rangle.\mathbf{0} \mid x^-!\langle \texttt{true} \rangle.\mathbf{0}) \quad \times$$

$$(\nu x)(x^-!\langle \texttt{false} \rangle.\mathbf{0} \mid x^+?(t : \texttt{Bool}).\mathbf{0} \mid x^+?(w : \texttt{Bool}).\mathbf{0}) \quad \times$$

$$(\nu x)(\nu y)(y^-!\langle 42 \rangle.x^+?(z : \texttt{Int}).\mathbf{0} \mid x^-!\langle 11 \rangle.y^+?(w : \texttt{Int}).\mathbf{0}) \quad \checkmark$$

$$(\nu x)(\nu y)(x^+?(z : \texttt{Int}).y^-!\langle 42 \rangle.\mathbf{0} \mid x^-!\langle 11 \rangle.y^+?(w : \texttt{Int}).\mathbf{0})$$

Exercise: Is it well typed?

$$(\boldsymbol{\nu} x)(x^+?(t : \texttt{Bool}).\mathbf{0} \mid x^-!\langle \texttt{true} \rangle.\mathbf{0}) \qquad \checkmark$$

$$(\boldsymbol{\nu} x)(x^+!\langle t \rangle.\mathbf{0} \mid x^-!\langle \texttt{true} \rangle.\mathbf{0}) \qquad \times$$

$$(\boldsymbol{\nu} x)(x^-!\langle \texttt{false} \rangle.\mathbf{0} \mid x^+?(t : \texttt{Bool}).\mathbf{0} \mid x^+?(w : \texttt{Bool}).\mathbf{0}) \qquad \times$$

$$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)(y^-!\langle 42 \rangle.x^+?(z : \texttt{Int}).\mathbf{0} \mid x^-!\langle 11 \rangle.y^+?(w : \texttt{Int}).\mathbf{0}) \qquad \checkmark$$

$$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)(x^+?(z : \texttt{Int}).y^-!\langle 42 \rangle.\mathbf{0} \mid x^-!\langle 11 \rangle.y^+?(w : \texttt{Int}).\mathbf{0}) \qquad \checkmark$$

Exercise: Is it well typed?

$$(\nu x)(x^+?(t : \texttt{Bool}).\mathbf{0} \mid x^-!\langle\texttt{true}\rangle.\mathbf{0}) \quad \checkmark$$

$$(\nu x)(x^+!\langle t\rangle.\mathbf{0} \mid x^-!\langle\texttt{true}\rangle.\mathbf{0}) \quad \times$$

$$(\nu x)(x^-!\langle\texttt{false}\rangle.\mathbf{0} \mid x^+?(t : \texttt{Bool}).\mathbf{0} \mid x^+?(w : \texttt{Bool}).\mathbf{0}) \quad \times$$

$$(\nu x)(\nu y)(y^-!\langle 42\rangle.x^+?(z : \texttt{Int}).\mathbf{0} \mid x^-!\langle 11\rangle.y^+?(w : \texttt{Int}).\mathbf{0}) \quad \checkmark$$

$$(\nu x)(\nu y)(x^+?(z : \texttt{Int}).y^-!\langle 42\rangle.\mathbf{0} \mid x^-!\langle 11\rangle.y^+?(w : \texttt{Int}).\mathbf{0}) \quad \checkmark$$

$$(\nu x)(x^- \triangleleft k.\mathbf{0} \mid x^+ \triangleright \{l_i : P_i\}_{i\in I}.\mathbf{0})$$

Exercise: Is it well typed?

$$(\nu x)(x^{+}?(t : \mathtt{Bool}).\mathbf{0} \mid x^{-}!\langle \mathtt{true} \rangle.\mathbf{0}) \qquad \checkmark$$

$$(\nu x)(x^{+}!\langle t \rangle.\mathbf{0} \mid x^{-}!\langle \mathtt{true} \rangle.\mathbf{0}) \qquad \times$$

$$(\nu x)(x^{-}!\langle \mathtt{false} \rangle.\mathbf{0} \mid x^{+}?(t : \mathtt{Bool}).\mathbf{0} \mid x^{+}?(w : \mathtt{Bool}).\mathbf{0}) \qquad \times$$

$$(\nu x)(\nu y)(y^{-}!\langle 42 \rangle.x^{+}?(z : \mathtt{Int}).\mathbf{0} \mid x^{-}!\langle 11 \rangle.y^{+}?(w : \mathtt{Int}).\mathbf{0}) \qquad \checkmark$$

$$(\nu x)(\nu y)(x^{+}?(z : \mathtt{Int}).y^{-}!\langle 42 \rangle.\mathbf{0} \mid x^{-}!\langle 11 \rangle.y^{+}?(w : \mathtt{Int}).\mathbf{0}) \qquad \checkmark$$

$$(\nu x)(x^{-} \triangleleft k.\mathbf{0} \mid x^{+} \triangleright \{l_i : P_i\}_{i \in I}.\mathbf{0}) \qquad \times$$

Outline

Outline

Research Timeline

Milner, Parrow, Walker 1989/1992

Honda 1993

Takeuchi, Honda, Kubo 1994

Honda, Vasconcelos, Kubo 1998

To be continued...

Milner 1993

Pierce, Sangiorgi 1993

Kobayashi, Pierce, Turner 1996

Sangiorgi 1998

On standard types for π- calculus

- $\sharp T$: channel used in input/output to transmit data of type T.

On standard types for π-calculus

- $\sharp T$: channel used in input/output to transmit data of type T.
- iT/oT: channel used *only* in input/output to transmit data of type T. [PS93]

On standard types for π- calculus

- $\sharp T$: channel used in input/output to transmit data of type T.
- iT/oT: channel used *only* in input/output to transmit data of type T. [PS93]
- $\ell_i T/\ell_o T$: channel used *only* in input/output and *exactly once* to transmit data of type T. [KPT96]

On standard types for π- calculus

- $\sharp T$: channel used in input/output to transmit data of type T.
- iT/oT: channel used *only* in input/output to transmit data of type T. [PS93]
- $\ell_i T/\ell_o T$: channel used *only* in input/output and *exactly once* to transmit data of type T. [KPT96]
- $\langle l_i : T_i \rangle_{i \in I}$: labelled disjoint union of types. [Sangio98]

Key words for standard π- types

For session-typed π- calculus:

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

Key words for standard π- types

For session-typed π- calculus:

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

1. Linearity forces a π channel to be used exactly once.
2. Capability of input/output of the same π channel split between two partners.
3. Restriction construct permits the creation of fresh private π channels.
4. Variant type permits choice.

Bridging the two worlds

To which extent session constructs are more complex and more expressive than the standard π- calculus constructs?

Research Timeline

Milner, Parrow, Walker 1989/1992

Honda 1993

Takeuchi, Honda, Kubo 1994

Honda, Vasconcelos, Kubo 1998

To be continued...

Milner 1993

Pierce, Sangiorgi 1993

Kobayashi, Pierce, Turner 1996

Sangiorgi 1998

Kobayashi 2007

Gay, Gesbert, Ravara 2008

Demangeon, Honda 2011

Dardha, Giachino, Sangiorgi 2012

Dardha 2014

Research Timeline

Milner, Parrow, Walker 1989/1992

Honda 1993	Milner 1993
↓	↓
Takeuchi, Honda, Kubo 1994	Pierce, Sangiorgi 1993
↓	↓
Honda, Vasconcelos, Kubo 1998	Kobayashi, Pierce, Turner 1996
↓	↓
To be continued...	Sangiorgi 1998

Kobayashi 2007

↓

Dardha, Giachino, Sangiorgi 2012

↓

Dardha 2014

Milner, Parrow and Walker 1989/1992
A calculus of mobile processes

Honda 1993
Types for dyadic interaction

Takeuchi, Honda and Kubo 1994
An interaction based language and its typing system

Honda, Vasconcelos and Kubo 1998
Language primitives and type discipline for structured communication-based programming

To be continued...

Milner 1993
The polyadic pi-calculus: a tutorial

Pierce and Sangiorgi 1993
Typing and subtyping for mobile processes

Kobayashi, Pierce and Turner 1996
Linearity and the pi-calculus

Sangiorgi 1998
An interpretation of typed objects into typed pi-calculus

Kobayashi 2007
Type systems for concurrent programs

Gay, Gesbert and Ravara 2008
Session types as generic process types

Demangeon and Honda 2011
Full abstraction in a subtyped pi-calculus with linear types

Dardha, Giachino and Sangiorgi 2012
Session types revisited

Dardha 2014
Recursive session types revisited

Key idea of the encoding

Encoding is based on:

1. Linearity of π- calculus channel types;
2. Input/Output channel capabilities;
3. Continuation-Passing principle.
4. Variant types for the π- calculus.

Intuition of the encoding

- Session types are encoded as linear channel types.
- ? and ! are encoded as ℓ_i and ℓ_o.
- $\&\{l_i : S_i\}_{i \in I}$ and $\oplus\{l_i : S_i\}_{i \in I}$ are encoded using *variant types*.
- Continuation of a session type becomes carried type.
- Dual operations in continuation become equal when carried.

Why is this interesting?

Benefits of the encoding:

1. Large reusability of standard typed π-calculus theory.
2. Derivation of properties for session π-calculus from the standard typed π-calculus. (e.g. SR, TS)
3. Elimination of redundancy in the syntax of types and terms and in the theory.
4. Encoding is robust (subtyping, polymorphism, higher-order).
5. Expressivity result for session types.

Encoding Finite Session Types: Example

Let

$$S = ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}$$

Then

$$[\![S]\!] = \ell_i[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Bool}, \emptyset[]]]]$$

Encoding Finite Session Types: Example

Let

$$S = ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}$$

Then

$$[\![S]\!] = \ell_i[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Bool}, \emptyset[]]]]$$

Encoding Finite Session Types: Example

Let

$$S = ?\texttt{Int}.?\texttt{Int}.!\texttt{Bool}.\texttt{end}$$

Then

$$[\![S]\!] = \ell_i[\texttt{Int}, \ell_i[\texttt{Int}, \ell_o[\texttt{Bool}, \emptyset[]]]]$$

Encoding Finite Session Types: Example

Let

$$S = \texttt{?Int.?Int.!Bool.end}$$

Then

$$[\![S]\!] = \ell_i[\texttt{Int}, \ell_i[\texttt{Int}, \ell_o[\texttt{Bool}, \emptyset[]]]]$$

Encoding Finite Session Types: Example

Let

$$S = \mathtt{?Int.?Int.!Bool.end}$$

Then

$$[\![S]\!] = \ell_i[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Bool}, \emptyset[]]]]$$

Encoding Finite Session Types: Example

Let

$$\overline{S} = !\texttt{Int}.!\texttt{Int}.?\texttt{Bool}.\texttt{end}$$

Then

$$[\![\overline{S}]\!] = \ell_o[\texttt{Int}, \ell_i[\texttt{Int}, \ell_o[\texttt{Bool}, \emptyset[]]]]$$

Remark

The encoding of dual types is as follows:

$$[\![S]\!] = \ell_i[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Bool}, \emptyset[]]]]$$

and

$$[\![\overline{S}]\!] = \ell_o[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Bool}, \emptyset[]]]]$$

Remark

The encoding of dual types is as follows:

$$[\![S]\!] = \ell_i[\texttt{Int}, \ell_i[\texttt{Int}, \ell_o[\texttt{Bool}, \emptyset[]]]]$$

and

$$[\![\overline{S}]\!] = \ell_o[\texttt{Int}, \ell_i[\texttt{Int}, \ell_o[\texttt{Bool}, \emptyset[]]]]$$

Remark

duality on session types boils down to opposite capabilities (i/o) of channel types, only in the outermost level!

Encoding of Session Types: Formally

$$
\begin{aligned}
[\![\mathtt{end}]\!] &\triangleq \emptyset[] \\
[\![!T.S]\!] &\triangleq \ell_o[[\![T]\!], [\![\overline{S}]\!]] \\
[\![?T.S]\!] &\triangleq \ell_i[[\![T]\!], [\![S]\!]] \\
[\![\oplus\{l_i : S_i\}_{i\in I}]\!] &\triangleq \ell_o[\langle l_i : [\![\overline{S_i}]\!]\rangle_{i\in I}] \\
[\![\&\{l_i : S_i\}_{i\in I}]\!] &\triangleq \ell_i[\langle l_i : [\![S_i]\!]\rangle_{i\in I}]
\end{aligned}
$$

Properties of the Encoding

Theorem

Encoding preserves typability of programs.

Theorem

Encoding preserves evaluation of programs.

Lemma

Encoding of dual session types gives dual linear π-types.

Outline

Propositions as Types[3]

propositions	as	types
proofs	as	programs
normalisation of proofs	as	evaluation of programs
Intuitionistic Natural Deduction	$\leftrightarrow$	Simply-Typed Lambda Calculus
Quantification over propositions	$\leftrightarrow$	Polymorphism
Modal Logical	$\leftrightarrow$	Monads (state, exceptions)

[3] I thank Phil Wadler for these two slides!

Propositions as Types

propositions	as	types
proofs	as	programs
normalisation of proofs	as	evaluation of programs
Intuitionistic Natural Deduction	$\leftrightarrow$	Simply-Typed Lambda Calculus
Quantification over propositions	$\leftrightarrow$	Polymorphism
Modal Logical	$\leftrightarrow$	Monads (state, exceptions)
???	$\leftrightarrow$	Process Calculus

Session Types and Linear Logic (1)

- What is the Curry-Howard correspondence for concurrency?

Session Types and Linear Logic (1)

- What is the Curry-Howard correspondence for concurrency?
- Caires & Pfenning (2010) established a correspondence between intuitionistic linear logic and session typed π- calculus.

Session Types and Linear Logic (1)

- What is the Curry-Howard correspondence for concurrency?
- Caires & Pfenning (2010) established a correspondence between intuitionistic linear logic and session typed π-calculus.
- Later on, Wadler (2012) established a correspondence between classical linear logic and session typed π-calculus.

Session Types and Linear Logic (2)

propositions	as	session types
proofs	as	π- processes
cut reduction	as	communication

Session Types and Classical Linear Logic (1)

- $A \parr B$ is interpreted as "input A then behave like B" $(?A.B)$

Session Types and Classical Linear Logic (1)

- $A \parr B$ is interpreted as "input A then behave like B" $(?A.B)$
- $A \otimes B$ is interpreted as "output A then behave like B" $(!A.B)$

Session Types and Classical Linear Logic (1)

- $A \parr B$ is interpreted as "input A then behave like B" ($?A.B$)
- $A \otimes B$ is interpreted as "output A then behave like B" ($!A.B$)
- $\&$ and $\oplus$ are interpreted as branch and select.

Session Types and Classical Linear Logic (1)

- $A \parr B$ is interpreted as "input A then behave like B" $(?A.B)$
- $A \otimes B$ is interpreted as "output A then behave like B" $(!A.B)$
- $\&$ and $\oplus$ are interpreted as branch and select.
- The correspondence has led to a re-examination of all aspects of session types, from a logical viewpoint.

Session Types and Classical Linear Logic (2)

Wadler 2012; Caires 2014 (@Luca Cardelli Fest)

(T-$\parr$)
$$\frac{P \vdash \Delta, y : A, x : B}{x?(y).P \vdash \Delta, x : A \parr B}$$

(T-$\otimes$)
$$\frac{P \vdash \Delta, y : A \qquad Q \vdash \Delta', x : B}{x!(y).(P \mid Q) \vdash \Delta, \Delta', x : A \otimes B}$$

(T-cut)
$$\frac{P \vdash \Delta, x : \overline{A} \qquad Q \vdash \Delta', x : A}{(\boldsymbol{\nu} x)(P \mid Q) \vdash \Delta, \Delta'}$$

(T-&)
$$\frac{P_i \vdash \Delta, x{:}A_i \qquad \forall i \in I}{x \triangleright \{l_i : P_i\}_{i \in I} \vdash \Delta, x{:}\&\{l_i : A_i\}_{i \in I}}$$

(T-$\oplus$)
$$\frac{P \vdash \Delta, x{:}A_j \qquad j \in I}{x \triangleleft l_j.P \vdash \Delta, x{:}\oplus\{l_i : A_i\}_{i \in I}}$$

Outline

Defining subtyping [4]

$$S \leq T$$

[4] I thank Luca Padovani for borrowing these two slides

Defining subtyping [4]

$$S \leq T$$

- Safe Substitutability (cf Liskov & 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.

[4] I thank Luca Padovani for borrowing these two slides

Defining subtyping [4]

$$S \leq T$$

- Safe Substitutability (cf Liskov & 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.
- Set Inclusion: in semantic subtyping (cf Castagna et al.)

$$[\![S]\!] \subseteq [\![T]\!]$$

[4] I thank Luca Padovani for borrowing these two slides

Defining subtyping [4]

$$S \leq T$$

- Safe Substitutability (cf Liskov & 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.
- Set Inclusion: in semantic subtyping (cf Castagna et al.)

$$[\![S]\!] \subseteq [\![T]\!]$$

- Property Preservation: (cf Liskov & Wing 1994)

$$\forall \phi.\ (\forall x : T.\ \phi(x)) \Longrightarrow (\forall y : S.\ \phi(y))$$

[4] I thank Luca Padovani for borrowing these two slides

Examples

- Set Inclusion:

$$\texttt{Even} \leq \texttt{Int} \text{ if and only if } [\![\texttt{Even}]\!] \leq [\![\texttt{Int}]\!]$$

- Property Preservation:

$$\{x : \texttt{Int}, y : \texttt{Int}, c : \texttt{Color}\} \leq \{x : \texttt{Int}, y : \texttt{Int}\}$$

  - ϕ(Point) = "Point has an x field".
  - ϕ(Point) = "Point has an y field".

Outline

The *Old* Maths Server and Client

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

$$S_{old} \triangleq \&\{\, add : ?\texttt{Int}.?\texttt{Int}.!\texttt{Int}.S_{old},\; neg : ?\texttt{Int}.!\texttt{Int}.S_{old}\; quit : \mathsf{end} \,\}$$

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

$$C_{old} \triangleq \oplus\{\, add : !\texttt{Int}.!\texttt{Int}.?\texttt{Int}.C_{old},\; neg : !\texttt{Int}.?\texttt{Int}.C_{old}\; quit : \mathsf{end} \,\}$$

The *New* Maths Server and Client

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

$$
\begin{aligned}
S_{new} \quad \triangleq \quad \&\{\ & \color{red}{mul} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.S_{new}, \\
& add : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.S_{new}, \\
& neg : ?\mathtt{Int}.!\mathtt{Int}.S_{new} \\
& quit : \mathrm{end}\ \}
\end{aligned}
$$

The *New* Maths Server and Client

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

$$
\begin{aligned}
S_{new} \quad \triangleq \quad \&\{\ & mul : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.S_{new}, \\
& add : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.S_{new}, \\
& neg : ?\mathtt{Int}.!\mathtt{Int}.S_{new} \\
& quit : \mathsf{end}\ \}
\end{aligned}
$$

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

$$
\begin{aligned}
C_{new} \quad \triangleq \quad \oplus\{\ & add : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Int}.C_{new}, \\
& quit : \mathsf{end}\ \}
\end{aligned}
$$

Subtyping: Channel Substitutability

- Gay & Hole, "*Subtyping for Session Types in the Pi Calculus*". ESOP 1999, Acta Informatica 2005.
- Allow interaction when the client does not know about all of the server's services.

$$\frac{I \subseteq J \qquad \forall i \in I.\ (S_i <: S'_i)}{\&\{l_i : S_i\}_{i \in I} <: \&\{l_j : S'_j\}_{j \in J}}$$

Subtyping: Channel Substitutability

- Gay & Hole, "*Subtyping for Session Types in the Pi Calculus*". ESOP 1999, Acta Informatica 2005.
- Allow interaction when the client does not know about all of the server's services.

$$\frac{I \subseteq J \qquad \forall i \in I.\ (S_i <: S'_i)}{\&\{l_i : S_i\}_{i \in I} <: \&\{l_j : S'_j\}_{j \in J}}$$

- Subtyping relation between S_{old} and S_{new}:

$$\begin{array}{rcl} S_{old} & = & \&\{add, neg, quit\} \\ S_{new} & = & \&\{mul, add, neg, quit\} \\ S_{old} & <: & S_{new} \end{array}$$

Subtyping: Channel Substitutability

- Gay & Hole, "*Subtyping for Session Types in the Pi Calculus*". ESOP 1999, Acta Informatica 2005.
- Allow interaction when the client does not know about all of the server's services.

$$\frac{I \subseteq J \qquad \forall i \in I.\ (S_i <: S'_i)}{\&\{l_i : S_i\}_{i \in I} <: \&\{l_j : S'_j\}_{j \in J}}$$

- Subtyping relation between S_{old} and S_{new}:

$$\begin{array}{rcl} S_{old} & = & \&\{add, neg, quit\} \\ S_{new} & = & \&\{mul, add, neg, quit\} \\ S_{old} & <: & S_{new} \end{array}$$

- Then the following holds:

From	$x : S_{new}$	$\vdash srv(x)$
we can conclude	$x : S_{old}$	$\vdash srv(x)$

Subtyping: Channel Substitutability

- Allow interaction when the client can choose from a smaller set choices than the ones offered by the server.

$$\frac{I \supseteq J \qquad \forall j \in J.\quad S_j <: S'_j}{\oplus\{l_i : S_i\}_{i\in I} <: \oplus\{l_j : S'_j\}_{j\in J}}$$

Subtyping: Channel Substitutability

- Allow interaction when the client can choose from a smaller set choices than the ones offered by the server.

$$\frac{I \supseteq J \qquad \forall j \in J. \quad S_j <: S'_j}{\oplus\{l_i : S_i\}_{i \in I} <: \oplus\{l_j : S'_j\}_{j \in J}}$$

- Subtyping relation between C_{old} and C_{new}:

$$\begin{array}{rcl} C_{old} & = & \oplus\{add, neg, quit\} \\ C_{new} & = & \oplus\{add, quit\} \\ C_{old} & <: & C_{new} \end{array}$$

Subtyping: Channel Substitutability

- Allow interaction when the client can choose from a smaller set choices than the ones offered by the server.

$$\frac{I \supseteq J \qquad \forall j \in J.\quad S_j <: S'_j}{\oplus\{l_i : S_i\}_{i \in I} <: \oplus\{l_j : S'_j\}_{j \in J}}$$

- Subtyping relation between C_{old} and C_{new}:

$$\begin{array}{rcl} C_{old} & = & \oplus\{add, neg, quit\} \\ C_{new} & = & \oplus\{add, quit\} \\ C_{old} & <: & C_{new} \end{array}$$

- Then the following holds:

From	$x : C_{new}$	$\vdash clt(x)$
we can conclude	$x : C_{old}$	$\vdash clt(x)$

Subtyping: Channel Substitutability

- Suppose that S_{old} has been published.

Subtyping: Channel Substitutability

- Suppose that S_{old} has been published.
- To use the server, a client creates a session channel c.

Subtyping: Channel Substitutability

- Suppose that S_{old} has been published.
- To use the server, a client creates a session channel c.
- The client sends $c^+ : S_{old}$ to the server, and keeps $c^- : \overline{S_{old}}$.

Subtyping: Channel Substitutability

- Suppose that S_{old} has been published.
- To use the server, a client creates a session channel c.
- The client sends $c^+ : S_{old}$ to the server, and keeps $c^- : \overline{S_{old}}$.
- The client is not aware that the server expects $x : S_{new}$.

Subtyping: Channel Substitutability

- Suppose that S_{old} has been published.
- To use the server, a client creates a session channel c.
- The client sends $c^+ : S_{old}$ to the server, and keeps $c^- : \overline{S_{old}}$.
- The client is not aware that the server expects $x : S_{new}$.
- Safe substitutability of channels: $S_{old} <: S_{new}$ and it is (semantically) safe for the server to be given $c^+ : S_{old}$.

Subtyping Rules for Session Types [GH05]

$$\frac{I \subseteq J \qquad \forall i \in I.\quad S_i <: S'_i}{\&\{l_i : S_i\}_{i \in I} <: \&\{l_j : S'_j\}_{j \in J}} \qquad \frac{I \supseteq J \qquad \forall j \in J.\quad S_j <: S'_j}{\oplus\{l_i : S_i\}_{i \in I} <: \oplus\{l_j : S'_j\}_{j \in J}}$$

Subtyping Rules for Session Types [GH05]

$$\frac{I \subseteq J \qquad \forall i \in I.\quad S_i <: S'_i}{\&\{l_i : S_i\}_{i\in I} <: \&\{l_j : S'_j\}_{j\in J}} \qquad \frac{I \supseteq J \qquad \forall j \in J.\quad S_j <: S'_j}{\oplus\{l_i : S_i\}_{i\in I} <: \oplus\{l_j : S'_j\}_{j\in J}}$$

$$\frac{}{end <: end} \qquad \frac{T <: T' \qquad S <: S'}{?T.S <: ?T'.S'} \qquad \frac{T' <: T \qquad S <: S'}{!T.S <: !T'.S'}$$

$?, \&$ are covariant

$!, \oplus$ are contravariant

Exercise: Which is subtype of which?

$![\mathtt{Even}].\mathtt{end}$	$![\mathtt{Int}].\mathtt{end}$
$![![\mathtt{Even}].\mathtt{end}].\mathtt{end}$	$![![\mathtt{Int}].\mathtt{end}].\mathtt{end}$
$?[![\mathtt{Even}].\mathtt{end}].\mathtt{end}$	$?[![\mathtt{Int}].\mathtt{end}].\mathtt{end}$
$\oplus\{\mathit{add} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}$	$\oplus\{\mathit{add} : \mathrm{end}, \mathit{neg} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}$
$![\oplus\{\mathit{add} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}]$	$![\oplus\{\mathit{add} : \mathrm{end}, \mathit{neg} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}]$
$\&\{\mathit{add} : \mathtt{Real}, \mathit{neg} : \mathtt{Int}\}$	$\&\{\mathit{add} : \mathtt{Int}, \mathit{neg} : \mathtt{Real}\}$

Exercise: Which is subtype of which?

$![\texttt{Even}].\texttt{end}$	$:>$	$![\texttt{Int}].\texttt{end}$
$![![\texttt{Even}].\texttt{end}].\texttt{end}$		$![![\texttt{Int}].\texttt{end}].\texttt{end}$
$?[![\texttt{Even}].\texttt{end}].\texttt{end}$		$?[![\texttt{Int}].\texttt{end}].\texttt{end}$
$\oplus\{\mathit{add} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}$		$\oplus\{\mathit{add} : \mathrm{end}, \mathit{neg} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}$
$![\oplus\{\mathit{add} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}]$		$![\oplus\{\mathit{add} : \mathrm{end}, \mathit{neg} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}]$
$\&\{\mathit{add} : \texttt{Real}, \mathit{neg} : \texttt{Int}\}$		$\&\{\mathit{add} : \texttt{Int}, \mathit{neg} : \texttt{Real}\}$

Exercise: Which is subtype of which?

$![\texttt{Even}].\texttt{end}$	$:>$	$![\texttt{Int}].\texttt{end}$
$![![\texttt{Even}].\texttt{end}].\texttt{end}$	$<:$	$![![\texttt{Int}].\texttt{end}].\texttt{end}$
$?[![\texttt{Even}].\texttt{end}].\texttt{end}$		$?[![\texttt{Int}].\texttt{end}].\texttt{end}$
$\oplus\{\mathit{add} : \text{end}, \mathit{quit} : \text{end}\}$		$\oplus\{\mathit{add} : \text{end}, \mathit{neg} : \text{end}, \mathit{quit} : \text{end}\}$
$![\oplus\{\mathit{add} : \text{end}, \mathit{quit} : \text{end}\}]$		$![\oplus\{\mathit{add} : \text{end}, \mathit{neg} : \text{end}, \mathit{quit} : \text{end}\}]$
$\&\{\mathit{add} : \texttt{Real}, \mathit{neg} : \texttt{Int}\}$		$\&\{\mathit{add} : \texttt{Int}, \mathit{neg} : \texttt{Real}\}$

Exercise: Which is subtype of which?

$![\mathtt{Even}].\mathtt{end}$	$:>$	$![\mathtt{Int}].\mathtt{end}$
$![![\mathtt{Even}].\mathtt{end}].\mathtt{end}$	$<:$	$![![\mathtt{Int}].\mathtt{end}].\mathtt{end}$
$?[![\mathtt{Even}].\mathtt{end}].\mathtt{end}$	$:>$	$?[![\mathtt{Int}].\mathtt{end}].\mathtt{end}$
$\oplus\{\mathit{add} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}$		$\oplus\{\mathit{add} : \mathrm{end}, \mathit{neg} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}$
$![\oplus\{\mathit{add} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}]$		$![\oplus\{\mathit{add} : \mathrm{end}, \mathit{neg} : \mathrm{end}, \mathit{quit} : \mathrm{end}\}]$
$\&\{\mathit{add} : \mathtt{Real}, \mathit{neg} : \mathtt{Int}\}$		$\&\{\mathit{add} : \mathtt{Int}, \mathit{neg} : \mathtt{Real}\}$

Exercise: Which is subtype of which?

$![\texttt{Even}].\texttt{end}$	$:>$	$![\texttt{Int}].\texttt{end}$
$![![\texttt{Even}].\texttt{end}].\texttt{end}$	$<:$	$![![\texttt{Int}].\texttt{end}].\texttt{end}$
$?[![\texttt{Even}].\texttt{end}].\texttt{end}$	$:>$	$?[![\texttt{Int}].\texttt{end}].\texttt{end}$
$\oplus\{\mathit{add} : \texttt{end}, \mathit{quit} : \texttt{end}\}$	$:>$	$\oplus\{\mathit{add} : \texttt{end}, \mathit{neg} : \texttt{end}, \mathit{quit} : \texttt{end}\}$
$![\oplus\{\mathit{add} : \texttt{end}, \mathit{quit} : \texttt{end}\}]$		$![\oplus\{\mathit{add} : \texttt{end}, \mathit{neg} : \texttt{end}, \mathit{quit} : \texttt{end}\}]$
$\&\{\mathit{add} : \texttt{Real}, \mathit{neg} : \texttt{Int}\}$		$\&\{\mathit{add} : \texttt{Int}, \mathit{neg} : \texttt{Real}\}$

Exercise: Which is subtype of which?

$![\mathtt{Even}].\mathtt{end}$	$:>$	$![\mathtt{Int}].\mathtt{end}$
$![![\mathtt{Even}].\mathtt{end}].\mathtt{end}$	$<:$	$![![\mathtt{Int}].\mathtt{end}].\mathtt{end}$
$?[![\mathtt{Even}].\mathtt{end}].\mathtt{end}$	$:>$	$?[![\mathtt{Int}].\mathtt{end}].\mathtt{end}$
$\oplus\{\mathit{add} : \mathtt{end}, \mathit{quit} : \mathtt{end}\}$	$:>$	$\oplus\{\mathit{add} : \mathtt{end}, \mathit{neg} : \mathtt{end}, \mathit{quit} : \mathtt{end}\}$
$![\oplus\{\mathit{add} : \mathtt{end}, \mathit{quit} : \mathtt{end}\}]$	$<:$	$![\oplus\{\mathit{add} : \mathtt{end}, \mathit{neg} : \mathtt{end}, \mathit{quit} : \mathtt{end}\}]$
$\&\{\mathit{add} : \mathtt{Real}, \mathit{neg} : \mathtt{Int}\}$		$\&\{\mathit{add} : \mathtt{Int}, \mathit{neg} : \mathtt{Real}\}$

Exercise: Which is subtype of which?

$![\mathtt{Even}].\mathtt{end}$	$:>$	$![\mathtt{Int}].\mathtt{end}$
$![![\mathtt{Even}].\mathtt{end}].\mathtt{end}$	$<:$	$![![\mathtt{Int}].\mathtt{end}].\mathtt{end}$
$?[![\mathtt{Even}].\mathtt{end}].\mathtt{end}$	$:>$	$?[![\mathtt{Int}].\mathtt{end}].\mathtt{end}$
$\oplus\{\mathit{add} : \mathtt{end}, \mathit{quit} : \mathtt{end}\}$	$:>$	$\oplus\{\mathit{add} : \mathtt{end}, \mathit{neg} : \mathtt{end}, \mathit{quit} : \mathtt{end}\}$
$![\oplus\{\mathit{add} : \mathtt{end}, \mathit{quit} : \mathtt{end}\}]$	$<:$	$![\oplus\{\mathit{add} : \mathtt{end}, \mathit{neg} : \mathtt{end}, \mathit{quit} : \mathtt{end}\}]$
$\&\{\mathit{add} : \mathtt{Real}, \mathit{neg} : \mathtt{Int}\}$	$\times$	$\&\{\mathit{add} : \mathtt{Int}, \mathit{neg} : \mathtt{Real}\}$

Subtyping: Process Substitutability

- Carbone, Honda & Yoshida (ESOP 2007);
  Demangeon & Honda (CONCUR 2011) define subtyping in the opposite direction: $S_{new} <: S_{old}$.

Subtyping: Process Substitutability

- Carbone, Honda & Yoshida (ESOP 2007); Demangeon & Honda (CONCUR 2011) define subtyping in the opposite direction: $S_{new} <: S_{old}$.
- They consider a session environment to be the type of a process:

$$x : S_{new} \vdash srv(x)$$

Subtyping: Process Substitutability

- Carbone, Honda & Yoshida (ESOP 2007);
  Demangeon & Honda (CONCUR 2011) define subtyping in the opposite direction: $S_{new} <: S_{old}$.
- They consider a session environment to be the type of a process:

$$x : S_{new} \vdash srv(x)$$

- They want safe substitutability of processes: the new server can be used in any context where an old server was expected.

Subtyping: Process Substitutability

- Carbone, Honda & Yoshida (ESOP 2007);
  Demangeon & Honda (CONCUR 2011) define subtyping in the opposite direction: $S_{new} <: S_{old}$.
- They consider a session environment to be the type of a process:

$$x : S_{new} \vdash srv(x)$$

- They want safe substitutability of processes: the new server can be used in any context where an old server was expected.
- Subsumption gives

$$x : S_{old} \vdash srv(x)$$

On the Subsumption Rule (1)

- Substitutability of channels:

$$\frac{\Gamma \vdash P \qquad \Gamma' <: \Gamma}{\Gamma' \vdash P}$$

- Example:

$$\frac{x : S_{new} \vdash srv(x) \qquad x : S_{old} <: x : S_{new}}{x : S_{old} \vdash srv(x)}$$

On the Subsumption Rule (2)

- Substitutability of processes:

$$\frac{\Gamma \vdash P \qquad \Gamma <: \Gamma'}{\Gamma' \vdash P}$$

- Example:

$$\frac{x : S_{new} \vdash srv(x) \qquad x : S_{new} <: x : S_{old}}{x : S_{old} \vdash srv(x)}$$

Simon J. Gay. "*Subtyping Supports Safe Session Substitution*". WadlerFest 2016

Outline

Subtyping Rules for Standard π- Types

$$\frac{}{T \leq T}\ (\mathrm{S}\pi\text{- }\textsc{Refl}) \qquad \frac{T \leq T' \qquad T' \leq T''}{T \leq T''}\ (\mathrm{S}\pi\text{- }\textsc{Trans})$$

$$\frac{\widetilde{T} \leq \widetilde{T'}}{\ell_i[\widetilde{T}] \leq \ell_i[\widetilde{T'}]}\ (\mathrm{S}\pi\text{- ii}) \qquad \frac{\widetilde{T'} \leq \widetilde{T}}{\ell_o[\widetilde{T}] \leq \ell_o[\widetilde{T'}]}\ (\mathrm{S}\pi\text{- oo})$$

$$\frac{I \subseteq J \qquad T_i \leq T'_j \quad \forall i \in I}{\langle l_i : T_i \rangle_{i \in I} \leq \langle l_j : T'_j \rangle_{j \in J}}\ (\mathrm{S}\pi\text{- }\textsc{Variant})$$

Subtyping

Theorem

For all session types S, S'. $S <: S'$ if and only if $[\![S]\!] \leq [\![S']\!]$.

Subtyping

Theorem

For all session types S, S'. $S <: S'$ if and only if $[\![S]\!] \leq [\![S']\!]$.

Derived from the encoding:

- Reflexivity and Transitivity of Subtyping.
- Lemmas (e.g., Substitution...) from the corresponding ones in the π- calculus. derived for free.

More on Subtyping

- Mostrous (2010) extended subtyping to allow some reordering of messages, when communication is asynchronous.

More on Subtyping

- Mostrous (2010) extended subtyping to allow some reordering of messages, when communication is asynchronous.
- Padovani (2011, 2013) has considered another form of subtyping, called fair subtyping.

More on Subtyping

- Mostrous (2010) extended subtyping to allow some reordering of messages, when communication is asynchronous.
- Padovani (2011, 2013) has considered another form of subtyping, called fair subtyping.
- Chen, Dezani & Yoshida (2014) have studied the preciseness of subtyping: the subtyping relation is sound and complete for safe substitutability.

Outline

Categorising language-based implementations of session types

- Binary vs. Multiparty
- Primitive vs. Library vs. External Tool
- Static vs. Dynamic vs. Hybrid checking

Programming Languages with Native BST: Static Typechecking

Sill:

- Functional programming language that supports session typed message passing concurrency.
- Based on the Curry-Howard correspondence of session types and intuitionistic linear logic (Caires & Pfenning 2010).
- Type preservation; deadlock and race freedom; support of subtyping, polymorphism and recursive types.
- Contributors: F. Pfenning, D. Griffith et al.

Programming Languages with Native BST: Static Typechecking

SePi:

- Concurrent, message-passing programming language based on the π-calculus.
- Based on synchronous, bidirectional channel based communication.
- Primitives for send/receive as well as offer/select choices.
- Contributors: J. Franco, V.Vasconcelos, D.Mostrous.

Programming Languages with Native BST: Static Typechecking[5]

Links:

- Programming language for web applications.
- Binary session types added as language primitives and statically typechecked.
- Developed at the University of Edinburgh.

[5] The following list of programming languages is taken from http://simonjf.com/2016/05/28/session-type-implementations.html

Mainstream Programming Languages with Binary Session Types

Haskell:

- effect-sessions: implementation in Concurrent Haskell; static typechecking. Orchard & Yoshida (POPL 2016)
- simple-session: la ibrary implementation of Haskell session types. Pucella & Tov (Haskell 2008)
- sessions: yet another embedding of session types in Haskell. Sackman & Eisenbach (TR 2008)

Mainstream Programming Languages with Binary Session Types

Java:

- CO2 Middleware: for Java applications, based on timed session types; dynamic monitoring for conformance of timing constraints.
  Bartoletti et al. (FACS 2015, FORTE 2015)

- (Eventful) Session Java: front-end and runtime library for Java; supports event-driven programming.
  Hu, Yoshida & Honda (ECOOP 2008);
  Hu et al. (ECOOP 2010)

Mainstream Programming Languages with Binary Session Types

Scala

- Based on the continuation-passing approach of Kobayashi 2007, and Dardha et al. 2012
- Message ordering is checked statically
- Linearity is checked dynamically.
- Scalas & Yoshida (ECOOP 2016)

Mainstream Programming Languages with Binary Session Types

OCaml: FuSe

- Lightweight implementation of BST in OCaml
- static check of message ordering and dynamic linearity check. Padovani 2015

Mainstream Programming Languages with Binary Session Types

OCaml: FuSe

- Lightweight implementation of BST in OCaml
- static check of message ordering and dynamic linearity check. Padovani 2015

Rust:

- Implementation of BST in Mozilla's Rust; use of Rust's affine type system. Jespersen, Munksgaard & Larsen (WGP 2015)

Outline

Multiparty Session Types (1)

- Binary session types can describe systems with multiple participants, but all protocols are pairwise and independent.

Multiparty Session Types (1)

- Binary session types can describe systems with multiple participants, but all protocols are pairwise and independent.
- Binary session types cannot constrain the order of two messages in different protocols.

Multiparty Session Types (1)

- Binary session types can describe systems with multiple participants, but all protocols are pairwise and independent.
- Binary session types cannot constrain the order of two messages in different protocols.
- Honda, Yoshida & Carbone (POPL 2008) developed a theory of multiparty session types.

Multiparty Session Types (1)

- Binary session types can describe systems with multiple participants, but all protocols are pairwise and independent.
- Binary session types cannot constrain the order of two messages in different protocols.
- Honda, Yoshida & Carbone (POPL 2008) developed a theory of multiparty session types.
- A global type specifies a multi-party protocol.

Multiparty Session Types (1)

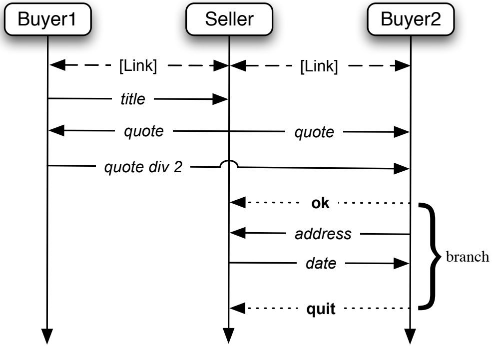
- Binary session types can describe systems with multiple participants, but all protocols are pairwise and independent.
- Binary session types cannot constrain the order of two messages in different protocols.
- Honda, Yoshida & Carbone (POPL 2008) developed a theory of multiparty session types.
- A global type specifies a multi-party protocol.
- A global type can be projected to local types, which specify the communication behaviour of each participant.

Multiparty Session Types (1)

- Binary session types can describe systems with multiple participants, but all protocols are pairwise and independent.
- Binary session types cannot constrain the order of two messages in different protocols.
- Honda, Yoshida & Carbone (POPL 2008) developed a theory of multiparty session types.
- A global type specifies a multi-party protocol.
- A global type can be projected to local types, which specify the communication behaviour of each participant.
- Local type checking guarantees communication safety.

Multiparty Session Types (2)

A buyer-seller example from Honda et al:



Multiparty Session Types (3)

The global type describes the whole protocol:

$$
\begin{array}{lll}
1. & \mathtt{B1} \to \mathtt{S}: & \mathsf{title}. \\
2. & \mathtt{S} \to \mathtt{B1}: & \mathsf{quote}. \\
3. & \mathtt{S} \to \mathtt{B2}: & \mathsf{quote}. \\
4. & \mathtt{B1} \to \mathtt{B2}: & \mathsf{quote}. \\
5. & \mathtt{B2} \to \mathtt{S}: & \left\{ \begin{array}{rll} \mathsf{ok}: & \mathtt{B2} \to \mathtt{S}: & \mathsf{address}. \\ & \mathtt{S} \to \mathtt{B2}: & \mathsf{date}.\mathsf{end}, \\ \mathsf{quit}: & \mathsf{end} & \end{array} \right\}
\end{array}
$$

Multiparty Session Types (4)

- Projection gives a local type for B1:

$$S!\mathsf{title}.S?\mathsf{quote}.B2!\mathsf{quote}$$

and for B2:

$$S?\mathsf{quote}.B1?\mathsf{quote}.S \oplus \{\mathsf{ok} : S!\mathsf{address}.S?\mathsf{date}.\mathsf{end},\ \mathsf{quit} : \mathsf{end}\}$$

Multiparty Session Types (4)

- Projection gives a local type for B1:

$$S!\mathrm{title}.S?\mathrm{quote}.B2!\mathrm{quote}$$

  and for B2:

$$S?\mathrm{quote}.B1?\mathrm{quote}.S \oplus \{\mathrm{ok} : S!\mathrm{address}.S?\mathrm{date}.\mathrm{end},\ \mathrm{quit} : \mathrm{end}\}$$

- Local type checking is similar to binary session types.

Multiparty Session Types (4)

- Projection gives a local type for B1:

$$S!\text{title}.S?\text{quote}.B2!\text{quote}$$

  and for B2:

$$S?\text{quote}.B1?\text{quote}.S \oplus \{\text{ok} : S!\text{address}.S?\text{date}.\text{end},\ \text{quit} : \text{end}\}$$

- Local type checking is similar to binary session types.
- Consistency conditions on the global type guarantee that the protocol can be realised by independent local participants.

Outline

Outline

Scribble

- Scribble is a language used to describe application-level protocols among communicating systems.

Scribble

- Scribble is a language used to describe application-level protocols among communicating systems.
- It is based on multiparty session types.

Scribble

- Scribble is a language used to describe application-level protocols among communicating systems.
- It is based on multiparty session types.
- Allows:
    - Specification of a protocol in the form of global session type;
    - Validation of the protocol;
    - Projection into the communicating participants, i.e., roles.

Scribble

- Scribble is a language used to describe application-level protocols among communicating systems.
- It is based on multiparty session types.
- Allows:
  - Specification of a protocol in the form of global session type;
  - Validation of the protocol;
  - Projection into the communicating participants, i.e., roles.
- Contributors: K.Honda, IC team (part of ABCD).

Scribble

- Scribble is a language used to describe application-level protocols among communicating systems.

- It is based on multiparty session types.

- Allows:
  - Specification of a protocol in the form of global session type;
  - Validation of the protocol;
  - Projection into the communicating participants, i.e., roles.

- Contributors: K.Honda, IC team (part of ABCD).

- Link: www.scribble.org

Scribble by example: The Bookstore Global Protocol

```
global protocol Bookstore(role Buyer1, role Buyer2,
    role Seller) {
  book(title) from Buyer1 to Seller;
  book(quote) from Seller to Buyer1, Buyer2;
  contribution(quote) from Buyer1 to Buyer2;
  choice at Buyer2 {
    ok from Buyer2 to Seller;
    deliver(address) from Buyer2 to Seller;
    deliver(date) from Seller to Buyer2;
   } or {
    quit from Buyer2 to Seller;
  }
}
```

The Bookstore Protocol: Buyer1

```
local protocol Bookstore_Buyer1(self Buyer1, role
    Buyer2, role Seller) {
  book(title) to Seller;
  book(quote) from Seller;
  contribution(quote) to Buyer2;
}
```

The Bookstore Protocol: Buyer2

```
local protocol Bookstore_Buyer2(role Seller, self
    Buyer2, role Buyer1) {
  book(quote) from Seller;
  contribution(quote) from Buyer1;
  choice at Buyer2{
    ok to Seller;
    deliver(address) to Seller;
    deliver(date) from Seller;
  } or {
    quit to Seller;
  }
}
```

Outline

Mungo

- Mungo is a Java front-end tool that statically checks the order of method calls.

Mungo

- Mungo is a Java front-end tool that statically checks the order of method calls.
- Based on the notions of session types and *typestate*, describing non-uniform objects.

Mungo

- Mungo is a Java front-end tool that statically checks the order of method calls.
- Based on the notions of session types and *typestate*, describing non-uniform objects.
- A Java class is annotated with a typestate. Mungo checks that method calls follow the declared typestate of an object.

Mungo

- Mungo is a Java front-end tool that statically checks the order of method calls.
- Based on the notions of session types and *typestate*, describing non-uniform objects.
- A Java class is annotated with a typestate. Mungo checks that method calls follow the declared typestate of an object.
- Contributors: ABCD Glasgow team.
  Based on Gay et al (POPL 2010);
  Kouzapas et al. (PPDP 2016)

The FileProtocol Example

```
typestate FileProtocol {
  Init = {
     Status open(): <OK: Open, ERROR: end>
  }
  Open = {
     BooleanEnum hasNext(): <TRUE: Read, FALSE: Close>,
     void close(): end
  }
  Read = {
     void read(): Open
  }
  Close = {
     void close(): end
  }
}
```

Outline

StMungo

- StMungo is a Java-based tool used to translate Scribble local protocols into typestate.

StMungo

- StMungo is a Java-based tool used to translate Scribble local protocols into typestate.
- After the translation, Mungo is used to statically typecheck the protocol.

StMungo

- StMungo is a Java-based tool used to translate Scribble local protocols into typestate.
- After the translation, Mungo is used to statically typecheck the protocol.
- Contributors: ABCD Glasgow team.
  Kouzapas et al. (PPDP 2016)

The Bookstore Protocol: Buyer2

```
local protocol Bookstore_Buyer2(role Seller, self
    Buyer2, role Buyer1) {
  book(quote) from Seller;
  contribution(quote) from Buyer1;
  choice at Buyer2{
    ok to Seller;
    deliver(address) to Seller;
    deliver(date) from Seller;
  } or {
    quit to Seller;
  }
}
```

The Buyer2 local protocol as Typestate

```
typestate Buyer2Protocol {
  State0 = {
    quote receive_quoteFromSeller(): State1
  }
  State1 = {
    quote receive_quoteFromBuyer1(): State2
  }
  State2 = {
    void send_OKToSeller(): State3,
    void send_QUITToSeller(): State5
  }
  State3 = {
     void send_addressToSeller(address): State4
  }
  State4 = {
    date receive_dateFromSeller(): end
  }
      ...
}
```

Outline

The SMTP Protocol: A Demo

Link: http://www.dcs.gla.ac.uk/research/mungo/

Mainstream Programming Languages with Multiparty Session Types[6]

Multiparty Session C:

- Static typechecking of MST in C
- Session communication happens via use of a library
- Ng, Yoshida & Honda (TOOLS 2012);
  Ng et al (HEART 2012)

DinGo Hunter

- External tool to statically analyse Go programs
- Static detection of deadlocks: extracting CFSMs and synthesising global graphs
- Ng & Yoshida (CC 2016)

[6] The following list of programming languages is taken from http://simonjf.com/2016/05/28/session-type-implementations.html

Mainstream Programming Languages with Multiparty Session Types

Session Actor

- A Python implementation for combining session types and the actor model of programming.
- Each actor may be involved in multiple roles, in multiple sessions.
- Communication is checked dynamically via compilation of Scribble protocols into CFSMs. Neykova & Yoshida (COORDINATION 2014)

Mainstream Programming Languages with Multiparty Session Types

Python

- SPY: implementation of MST in Python using runtime monitoring. Neykova (PLACES 2013); Neykova, Yoshida & Hu (RV 2013); Hu et al (RV 2013)

Erlang

- Dynamic monitoring of communication (MST) for Erlang applications
- Inspired by Session Actor. Simon Fowler (MSc thesis, 2015)

Outline

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 π)
- Lock Freedom: communications eventually succeed *even if* the whole process diverges. (Standard π)
- Progress: In-session communications eventually succeed, provided that a suitable context can be found. (Session π)

Deadlock Freedom vs. Lock Freedom

- Consider the process:

$$P = (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)\big(\ x^{+}?(z).y^{+}!\langle z\rangle \ \mid\ y^{-}?(w).x^{-}!\langle w\rangle\ \big)$$

It is deadlocked and hence locked!

Deadlock Freedom vs. Lock Freedom

- Consider the process:

$$P = (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)\big(\ x^{+}?(z).y^{+}!\langle z\rangle \ \mid\ y^{-}?(w).x^{-}!\langle w\rangle\ \big)$$

  It is deadlocked and hence locked!

- Consider the process:

$$Q = (\boldsymbol{\nu} x)(x^{+}?(z) \mid \Omega)$$

  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 & Montesi 2014)

More Advanced Topics in Sessions

Session types theories include notions and studies in the following.

- Notions of subtyping, polymorphism, higher-order.
- Study of liveness properties: deadlock freedom, lock freedom and progress.
- Asynchrony and synchrony.
- Static typechecking and dynamic monitoring.
- Finite and recursive session types.
- Study of security (e.g., information flow).
- Exceptions, time-outs.
- Point-to-point and broadcasting
- And many more...

Conclusions

- Session Types are a very simple but powerful formalism to model protocols in distributed systems.
- Developed for calculi as well as programming languages and various paradigms.
- Many interesting features.
- Part of behavioural types, including also contracts, typestates...

$$
\begin{array}{l}
\mathrm{Audience}!\langle \mathrm{ThankYou} \rangle. \\
\mathrm{rec}\ X\{\ \&\ \{ \\
\qquad\qquad \mathit{more} : \mathrm{Audience}?(y : \mathrm{Question}).\mathrm{Audience}!\langle \mathrm{Answer} \rangle.X, \\
\qquad\qquad \mathit{quit} : \mathrm{end}\} \\
\}
\end{array}
$$

References I

Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida.
Global progress in dynamically interleaved multiparty sessions.
In *CONCUR*, pages 418–433, 2008.

Marco Carbone and Søren Debois.
A graphical approach to progress for structured communication in web services.
In *ICE*, pages 13–27, 2010.

Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida.
On the preciseness of subtyping in session types.
In *PPDP*. ACM, 2014.

References II

Marco Carbone, Ornela Dardha, and Fabrizio Montesi.
Progress as compositional lock-freedom.
In *COORDINATION*, volume 8459 of *LNCS*, pages 49–64. Springer, 2014.

Luís Caires and Frank Pfenning.
Session types as intuitionistic linear propositions.
In *CONCUR*, pages 222–236, 2010.

Ornela Dardha, Elena Giachino, and Davide Sangiorgi.
Session types revisited.
In *PPDP*, pages 139–150, New York, NY, USA, 2012. ACM.

Romain Demangeon and Kohei Honda.
Full abstraction in a subtyped pi-calculus with linear types.
In *CONCUR*, pages 280–296, 2011.

References III

Simon J. Gay.
Subtyping supports safe session substitution.
In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, volume 9600 of *LNCS*, pages 95–108. Springer, 2016.

Simon J. Gay and Malcolm Hole.
Subtyping for session types in the pi calculus.
Acta Informatica, 42(2-3):191–225, nov 2005.

Kohei Honda, Nobuko Yoshida, and Marco Carbone.
Multiparty asynchronous session types.
In *POPL*, volume 43(1), pages 273–284. ACM, 2008.

Naoki Kobayashi.
A type system for lock-free processes.
Inf. Comput., 177(2):122–159, 2002.

References IV

Naoki Kobayashi.
A new type system for deadlock-free processes.
In *CONCUR*, pages 233–247, 2006.

Luca Padovani.
Fair subtyping for multi-party session types.
In *COORDINATION*, pages 127–141, Berlin, Heidelberg, 2011. Springer-Verlag.

Luca Padovani.
Fair subtyping for open session types.
In *ICALP*, 2013.

Vasco T. Vasconcelos.
Fundamentals of session types.
Information Computation, 217:52–70, 2012.

References V

Philip Wadler.
Propositions as sessions.
In *ICFP*, pages 273–286, 2012.