Session Types Revisited

Ornela Dardha[1]
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

[1]Supported 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
  - π-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/offer/external choice;
- $\oplus$: select/internal choice;
- $?\texttt{Int}.T$: input `Int`, continue as T;
- $!\texttt{Int}.T$: output `Int`, continue as T;
- "$\cdot$" indicates sequencing;
- *add, neg, quit*: choice labels, all different;
- `end` marks the end of the protocol.

The Maths Server and Client: Types / Protocols

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

$$
\begin{aligned}
S \quad \triangleq \quad \&\{ \quad & \mathit{plus} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.\mathtt{end}, \\
& \mathit{equal} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}, \\
& \mathit{neg} : ?\mathtt{Int}.!\mathtt{Int}.\mathtt{end} \quad \}
\end{aligned}
$$

The Maths Server and Client: Types / Protocols

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

$$S \triangleq \&\{ \begin{array}{l} \mathit{plus} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.\mathtt{end}, \\ \mathit{equal} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}, \\ \mathit{neg} : ?\mathtt{Int}.!\mathtt{Int}.\mathtt{end} \quad \} \end{array}$$

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

$$C \triangleq \oplus\{ \begin{array}{l} \mathit{plus} : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Int}.\mathtt{end}, \\ \mathit{equal} : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Bool}.\mathtt{end}, \\ \mathit{neg} : !\mathtt{Int}.?\mathtt{Int}.\mathtt{end} \quad \} \end{array}$$

The Maths Server and Client: Types / Protocols

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

$$S \triangleq \&\{ \begin{array}{l} \mathit{plus} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.\mathtt{end}, \\ \mathit{equal} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}, \\ \mathit{neg} : ?\mathtt{Int}.!\mathtt{Int}.\mathtt{end} \quad \} \end{array}$$

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

$$C \triangleq \oplus\{ \begin{array}{l} \mathit{plus} : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Int}.\mathtt{end}, \\ \mathit{equal} : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Bool}.\mathtt{end}, \\ \mathit{neg} : !\mathtt{Int}.?\mathtt{Int}.\mathtt{end} \quad \} \end{array}$$

Type duality: $S = \overline{C}$

The Maths Server: Process and Type

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

$$\begin{array}{rcll} \mathit{server} & \triangleq & x \triangleright \{ & \mathit{plus} : x?(v_1).x?(v_2).x!\langle v_1 + v_2 \rangle.\mathbf{0}, \\ & & & \mathit{equal} : x?(v_1).x?(v_2).x!\langle v_1 == v_2 \rangle.\mathbf{0}, \\ & & & \mathit{neg} : x?(v).x!\langle -v \rangle.\mathbf{0} \quad \} \end{array}$$

Session endpoint x obeys:

$$\begin{array}{rcll} S & \triangleq & \&\{ & \mathit{plus} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.\mathtt{end}, \\ & & & \mathit{equal} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}, \\ & & & \mathit{neg} : ?\mathtt{Int}.!\mathtt{Int}.\mathtt{end} \quad \} \end{array}$$

The Maths Client: Process and Type

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

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

Session endpoint y obeys:

$$\begin{aligned} C \triangleq \oplus\{ & plus : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Int}.\mathtt{end}, \\ & equal : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Bool}.\mathtt{end}, \\ & neg : !\mathtt{Int}.?\mathtt{Int}.\mathtt{end} \quad \} \end{aligned}$$

Client/Server Interaction (π-calculus OS)

$$(\nu xy : C)(\mathit{server} \mid \mathit{client})$$

Client/Server Interaction (π-calculus OS)

$$(\nu xy : C)(\mathit{server} \mid \mathit{client})$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Bool}.\mathtt{end})\big(x?(v_1).x?(v_2).x!\langle v_1 == v_2\rangle.\mathbf{0} \mid y!\langle 3\rangle.y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$

Client/Server Interaction (π-calculus OS)

$$(\nu xy : C)(\mathit{server} \mid \mathit{client})$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Bool}.\mathtt{end})\big(x?(v_1).x?(v_2).x!\langle v_1 == v_2\rangle.\mathbf{0} \mid y!\langle 3\rangle.y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : !\mathtt{Int}.?\mathtt{Bool}.\mathtt{end})\big(x?(v_2).x!\langle 3 == v_2\rangle.\mathbf{0} \mid y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$

Client/Server Interaction (π-calculus OS)

$$(\nu xy : C)(server \mid client)$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : !\texttt{Int}.!\texttt{Int}.?\texttt{Bool}.\texttt{end})\big(x?(v_1).x?(v_2).x!\langle v_1 == v_2\rangle.\mathbf{0} \mid y!\langle 3\rangle.y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : !\texttt{Int}.?\texttt{Bool}.\texttt{end})\big(x?(v_2).x!\langle 3 == v_2\rangle.\mathbf{0} \mid y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : ?\texttt{Bool}.\texttt{end})\big(x!\langle 3 == 5\rangle.\mathbf{0} \mid y?(eq).\mathbf{0}\big)$$

Client/Server Interaction (π-calculus OS)

$$(\nu xy : C)(server \mid client)$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : !\texttt{Int}.!\texttt{Int}.?\texttt{Bool}.\texttt{end})\big(x?(v_1).x?(v_2).x!\langle v_1 == v_2\rangle.\mathbf{0} \mid y!\langle 3\rangle.y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : !\texttt{Int}.?\texttt{Bool}.\texttt{end})\big(x?(v_2).x!\langle 3 == v_2\rangle.\mathbf{0} \mid y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : ?\texttt{Bool}.\texttt{end})\big(x!\langle 3 == 5\rangle.\mathbf{0} \mid y?(eq).\mathbf{0}\big)$$

$$\downarrow$$

$$(\boldsymbol{\nu} xy : \texttt{end})(\mathbf{0} \mid \mathbf{0})$$

Client/Server Interaction (π-calculus OS)

$$(\nu xy : C)(server \mid client)$$
$$\downarrow$$
$$(\boldsymbol{\nu} xy : !\texttt{Int}.!\texttt{Int}.?\texttt{Bool}.\texttt{end})\big(x?(v_1).x?(v_2).x!\langle v_1 == v_2\rangle.\mathbf{0} \mid y!\langle 3\rangle.y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$
$$\downarrow$$
$$(\boldsymbol{\nu} xy : !\texttt{Int}.?\texttt{Bool}.\texttt{end})\big(x?(v_2).x!\langle 3 == v_2\rangle.\mathbf{0} \mid y!\langle 5\rangle.y?(eq).\mathbf{0}\big)$$
$$\downarrow$$
$$(\boldsymbol{\nu} xy : ?\texttt{Bool}.\texttt{end})\big(x!\langle 3 == 5\rangle.\mathbf{0} \mid y?(eq).\mathbf{0}\big)$$
$$\downarrow$$
$$(\boldsymbol{\nu} xy : \texttt{end})(\mathbf{0} \mid \mathbf{0})$$
$$\equiv$$
$$\mathbf{0}$$

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

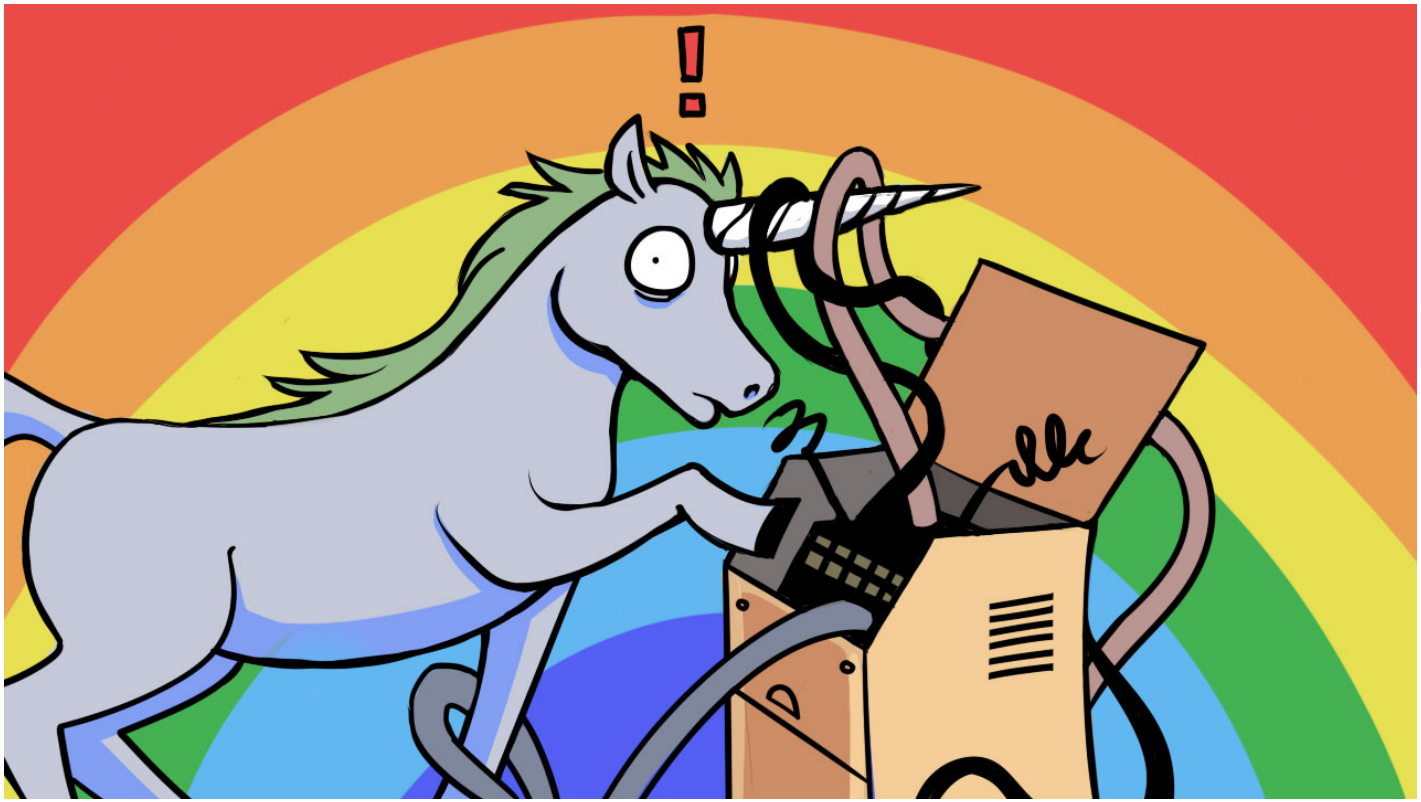
- Communication Safety: expected types of exchanged data.
- Session Fidelity: expected structure of session channels.
- Privacy: session channel owned *only* by communicating peers.

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 π-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
	$\sharp T$	standard channel type
	$\mathtt{Bool}$	boolean type
	$\ldots$	other type constructs

The Session π-Calculus: Terms

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

Session Typing Rules

(T-Res)

$$\frac{\Gamma, x : S, y : \overline{S} \vdash P}{\Gamma \vdash (\nu xy)P}$$

(T-In)

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

(T-Out)

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

Standard Typing Rules

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

$$\frac{\Gamma, x : \sharp T, y : T \vdash P}{\Gamma, x : \sharp T \vdash x?(y).P} \quad \text{(T-StndIn)}$$

$$\frac{\Gamma_1, x : \sharp T \vdash P \quad \Gamma_2 \vdash v : T}{(\Gamma_1, x : \sharp T) + \Gamma_2 \vdash x!\langle v \rangle.P} \quad \text{(T-StndOut)}$$

From Session Types to Standard π-Calculus Types

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

On standard types for π-calculus

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

On standard types for π-calculus

- $\sharp T$: 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)

On standard types for π-calculus

- $\sharp T$: 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)

On standard types for π-calculus

- $\sharp T$: 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 π-types

Keywords for session types:

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

Session types vs. standard π-types

Keywords for session types:

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

Keywords for standard π-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 π-Calculus: Types

$\tau ::=$	$\ell_i[\widetilde{\tau}]$	linear input
	$\ell_o[\widetilde{\tau}]$	linear output
	$\ell_\sharp[\widetilde{\tau}]$	linear connection
	$\emptyset[]$	channel with no capability
	$\langle l_i : \tau_i \rangle_{i \in I}$	variant type
	$\sharp\tau$	standard channel type
	`Bool`	boolean type
	$\cdots$	other constructs

The Standard π-Calculus: Terms

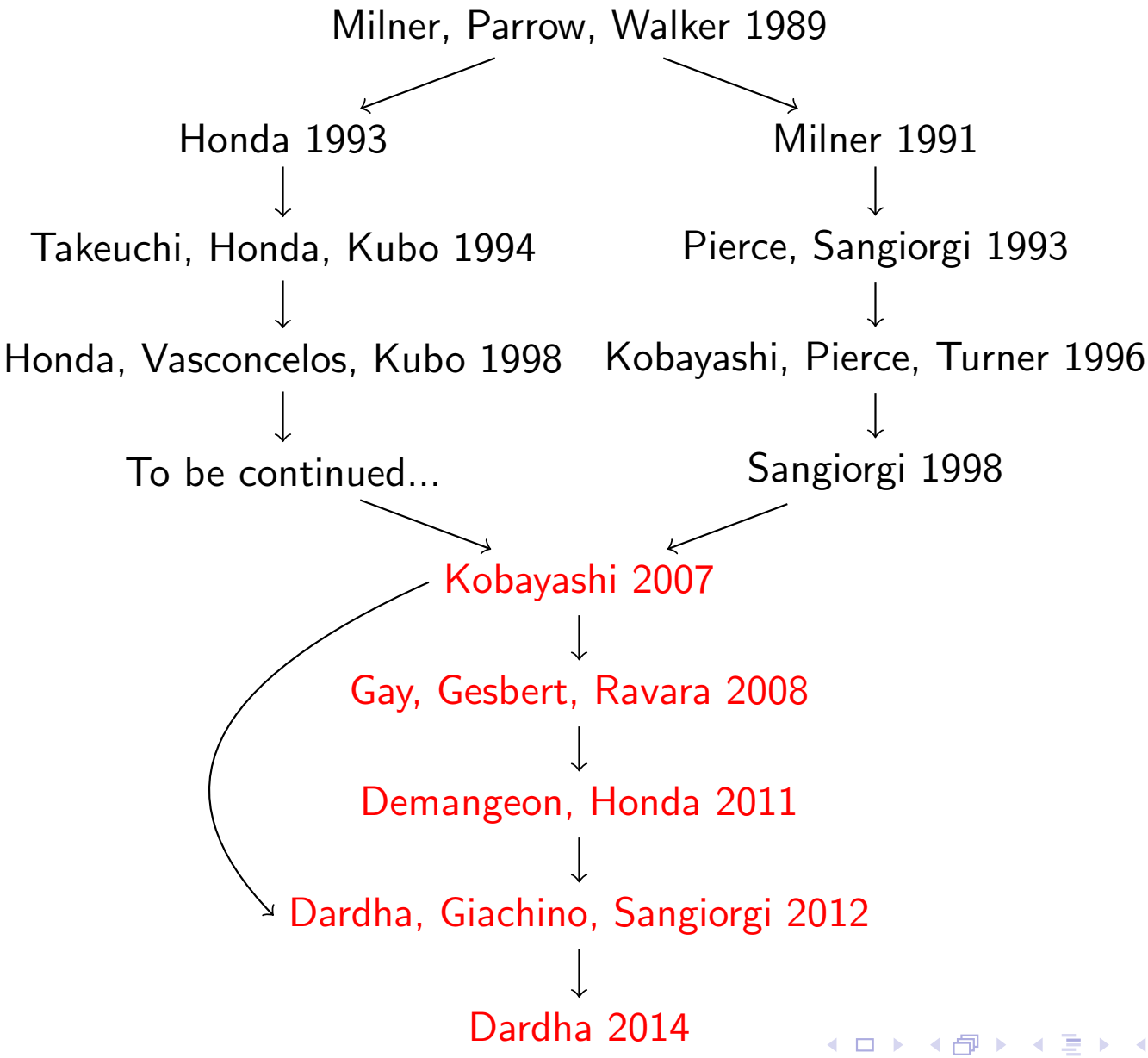
$P, Q ::=$	$\mathbf{0}$	inaction
	$x!\langle\tilde{v}\rangle.P$	output
	$x?(\tilde{y}).P$	input
	$P \mid Q$	composition
	$(\boldsymbol{\nu} x)P$	channel restriction
	$\mathbf{case}\ v\ \mathbf{of}\ \{l_i\_x_i \triangleright P_i\}_{i \in I}$	case process

$v ::=$	x	variable
	b	boolean values
	l_v	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



Research Timeline

Milner, Parrow, Walker 1989

Honda 1993

Milner 1991

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

Session Types Revisited

Session types are encoded into standard π-calculus types.

Encoding is based on:

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

Intuition of the Encoding

1. Session types encoded as linear π-calculus channel types.
2. ? and ! encoded as ℓ_i and ℓ_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 π-calculus theory to session types.
3. Derivation of properties for session π-calculus from the standard typed π-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

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

Then

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

Encoding Session Types: Example

Let

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

Then

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

Encoding Session Types: Example

Let

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

Then

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

Encoding Session Types: Example

Let

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

Then

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

Encoding Session Types: Example

Let

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

Then

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

Encoding Session Types: Example

Let

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

Then

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

What happens to duality?

The encoding of dual types is as follows:

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

and

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

What happens to duality?

The encoding of dual types is as follows:

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

and

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

Remark

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

Encoding the Maths Server and Client: Types

$$
\begin{aligned}
S \;\triangleq\; \&\{ \;& \mathit{plus} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.\mathtt{end}, \\
& \mathit{equal} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}, \\
& \mathit{neg} : ?\mathtt{Int}.!\mathtt{Int}.\mathtt{end} \;\}
\end{aligned}
$$

$$
\begin{aligned}
[\![S]\!] \;=\; \ell_{\mathsf{i}}[\langle \;& \mathit{plus}\_\ell_i[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Int}, \emptyset[]]]] \\
& \mathit{equal}\_\ell_i[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Bool}, \emptyset[]]]], \\
& \mathit{neg}\_\ell_i[\mathtt{Int}, \ell_o[\mathtt{Int}, \emptyset[]]] \;\rangle]
\end{aligned}
$$

Encoding the Maths Server and Client: Types

$$
\begin{aligned}
C \;\triangleq\; \oplus\{ \;& \mathit{plus} : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Int}.\mathtt{end}, \\
& \mathit{equal} : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Bool}.\mathtt{end}, \\
& \mathit{neg} : !\mathtt{Int}.?\mathtt{Int}.\mathtt{end} \quad \}
\end{aligned}
$$

$$
\begin{aligned}
[\![C]\!] \;=\; \ell_o[\langle \;& \mathit{plus}\_\ell_i[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Int}, \emptyset[]]]] \\
& \mathit{equal}\_\ell_i[\mathtt{Int}, \ell_i[\mathtt{Int}, \ell_o[\mathtt{Bool}, \emptyset[]]]], \\
& \mathit{neg}\_\ell_i[\mathtt{Int}, \ell_o[\mathtt{Int}, \emptyset[]]] \quad \rangle]
\end{aligned}
$$

Encoding of Session Types: Formally

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

Encoding the Maths Server and Client: Processes

Renaming function is the identity function $\emptyset$:

$$\begin{aligned}
[\![(\boldsymbol{\nu} xy)(\mathit{server} \mid \mathit{client})]\!]_{\emptyset} &= (\boldsymbol{\nu} z)[\![(\mathit{server} \mid \mathit{client})]\!]_{\{x,y \mapsto z\}} \\
&= (\boldsymbol{\nu} z)\big([\![\mathit{server}]\!]_{\{x \mapsto z\}} \mid [\![\mathit{client}]\!]_{\{y \mapsto z\}}\big)
\end{aligned}$$

$\{x, y \mapsto z\}$ maps x and y to a fresh name z;

Encoding the Maths Server and Client: Processes

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

$$
\begin{aligned}
&[\![client]\!]_{\{y \mapsto z\}} = \\
&\quad (\boldsymbol{\nu} s)z!\langle equal\_s\rangle.(\boldsymbol{\nu} c)s!\langle 3, c\rangle.(\boldsymbol{\nu} c')c!\langle 5, c'\rangle.c'?(eq, c'').\mathbf{0}
\end{aligned}
$$

Encoding of Session Terms: Formally

$$
\begin{aligned}
[\![x!\langle v\rangle.P]\!]_f &\triangleq (\boldsymbol{\nu} c) f_x!\langle [\![v]\!]_f, c\rangle.[\![P]\!]_{f,\{x\mapsto c\}} \\
[\![x?(y).P]\!]_f &\triangleq f_x?(y,c).[\![P]\!]_{f,\{x\mapsto c\}} \\
[\![x \triangleleft l_j.P]\!]_f &\triangleq (\boldsymbol{\nu} c) f_x!\langle l_j\_c\rangle.[\![P]\!]_{f,\{x\mapsto c\}} \\
[\![x \triangleright \{l_i : P_i\}_{i\in I}]\!]_f &\triangleq f_x?(y).\ \textbf{case}\ y\ \textbf{of}\ \{l_i\_c \triangleright [\![P_i]\!]_{f,\{x\mapsto c\}}\}_{i\in I} \\
[\![P \mid Q]\!]_f &\triangleq [\![P]\!]_f \mid [\![Q]\!]_f \\
[\![(\boldsymbol{\nu} xy)P]\!]_f &\triangleq (\boldsymbol{\nu} z)[\![P]\!]_{f,\{x,y\mapsto z\}}
\end{aligned}
$$

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.

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)

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 π-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 π-calculus types: compilation of session types into an "assembly" language.
- Encoding proved faithful: derive properties of session from the corresponding ones of standard π-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.

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

Milner, Parrow and Walker 1989
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 1991
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

Extensions of the Encoding of Sessions

Subtyping

$$S \leq T$$

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:

$$S_{old} \triangleq \&\{ \; \mathit{plus} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.\mathtt{end}, \; \mathit{equal} : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}, \; \mathit{neg} : ?\mathtt{Int}.!\mathtt{Int}.\mathtt{end} \; \}$$

Updating the Maths Server by Subtyping

The old session type of the server's endpoint:

$$S_{old} \triangleq \&\{ \begin{array}{l} plus : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.\mathtt{end}, \\ equal : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}, \\ neg : ?\mathtt{Int}.!\mathtt{Int}.\mathtt{end} \quad \} \end{array}$$

The new session type of the server's endpoint:

$$S_{new} \triangleq \&\{ \begin{array}{l} mul : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}..\mathtt{end}, \\ plus : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Int}.\mathtt{end}, \\ equal : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}, \\ neg : ?\mathtt{Int}.!\mathtt{Int}.\mathtt{end} \quad \} \end{array}$$

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 $[\![S]\!] \leq [\![S']\!]$.

Subtyping by Encoding

Theorem

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

Derived from the encoding and subtyping in standard typed π-calculus:

- Reflexivity and transitivity of subtyping for sessions.
- Lemmas (e.g., Substitution...) from the corresponding ones in the π-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 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:

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

It is deadlocked and hence locked!

Deadlock Freedom vs. Lock Freedom

- Consider the process:

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

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 π-calculus.
- We use our encoding of session π-calculus to the standard typed π-calculus.

Typing Progress

```
procedure PROGRESS(Γ, P)
    Check Γ ⊢ P
    Build close(P) from Γ
    Encode ⟦close(P)⟧_f = P′
    return TyPiCal(P′)
end procedure
```

References I

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

Ornela Dardha.
Recursive Session Types Revisited.
In *the Third Workshop on Behavioural Types, BEAT*, volume 162 of *EPTCS*, pages 27–34, 2014.

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

References II

Ornela Dardha.
Type Systems for Distributed Programs: Components and Sessions, volume 7 of *Atlantis Studies in Computing*.
Atlantis Press, July 2016.

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.
Acta Informatica, 42(2-3):191–225, nov 2005.

References III

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

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

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

Naoki Kobayashi.
Type systems for concurrent programs.
In *10th Anniversary Colloquium of UNU/IIST*, pages 439–453, 2002.

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.
In *ECOOP*, volume 56 of *LIPIcs*, pages 21:1–21:28, 2016.