Recursive Session Types Revisited

Ornela Dardha

School of Computing Science
University of Glasgow

September 1, 2014

Gentle Intro

- Session types are a type formalism used to model structured communication-based programming.
- Suitable for designing protocols in a concurrent and distributed scenario.
- Guarantee privacy, communication safety and session fidelity.

Session Types in Practice: Equality Test

$$\text{server} \stackrel{\text{def}}{=} x_1?(v).x_1?(w).x_1!\langle v == w \rangle.\mathbf{0}$$

$$\text{client} \stackrel{\text{def}}{=} x_2!\langle 3 \rangle.x_2!\langle 5 \rangle.x_2?(eq).\mathbf{0}$$

The system is given by

$$(\boldsymbol{\nu} x_1 x_2)\ \big(\text{server} \mid \text{client}\big)$$

Session Types in Practice: Equality Test

$$\text{server} \stackrel{\texttt{def}}{=} x_1?(v).x_1?(w).x_1!\langle v == w\rangle.\mathbf{0}$$

$$\text{client} \stackrel{\texttt{def}}{=} x_2!\langle 3\rangle.x_2!\langle 5\rangle.x_2?(eq).\mathbf{0}$$

The system is given by

$$(\boldsymbol{\nu} x_1 x_2)\ \big(\text{server} \mid \text{client}\big)$$

Where

$$x_1 : ?\texttt{Int}.?\texttt{Int}.!\texttt{Bool}.\texttt{end}$$

and

$$x_2 : !\texttt{Int}.!\texttt{Int}.?\texttt{Bool}.\texttt{end}$$

Session Types in Practice: Equality Test

$$\text{server} \stackrel{\text{def}}{=} x_1?(v).x_1?(w).x_1!\langle v == w\rangle.\mathbf{0}$$

$$\text{client} \stackrel{\text{def}}{=} x_2!\langle 3\rangle.x_2!\langle 5\rangle.x_2?(eq).\mathbf{0}$$

The system is given by

$$(\boldsymbol{\nu} x_1 x_2)\ \big(\text{server} \mid \text{client}\big)$$

Where

$$x_1 : ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathtt{end}$$

and

$$x_2 : !\mathtt{Int}.!\mathtt{Int}.?\mathtt{Bool}.\mathtt{end}$$

Session Types in Practice: Equality Test

$$\text{server} \stackrel{\texttt{def}}{=} x_1?(v).x_1?(w).x_1!\langle v == w\rangle.\mathbf{0}$$

$$\text{client} \stackrel{\texttt{def}}{=} x_2!\langle 3\rangle.x_2!\langle 5\rangle.x_2?(eq).\mathbf{0}$$

The system is given by

$$(\boldsymbol{\nu} x_1 x_2)\ \big(\text{server} \mid \text{client}\big)$$

Where

$$x_1 : \texttt{?Int.?Int.!Bool.end}$$

and

$$x_2 : \texttt{!Int.!Int.?Bool.end}$$

Session Types in Practice: Equality Test

$$\text{server} \stackrel{\text{def}}{=} x_1?(v).x_1?(w).x_1!\langle v == w\rangle.\mathbf{0}$$

$$\text{client} \stackrel{\text{def}}{=} x_2!\langle 3\rangle.x_2!\langle 5\rangle.x_2?(eq).\mathbf{0}$$

The system is given by

$$(\boldsymbol{\nu} x_1 x_2)\ \big(\text{server} \mid \text{client}\big)$$

Where

$$x_1 : \texttt{?Int.?Int.!Bool.end}$$

and

$$x_2 : \texttt{!Int.!Int.?Bool.end}$$

Session Types in Practice: Equality Test

$$\text{server} \stackrel{\text{def}}{=} x_1?(v).x_1?(w).x_1!\langle v == w\rangle.\mathbf{0}$$

$$\text{client} \stackrel{\text{def}}{=} x_2!\langle 3\rangle.x_2!\langle 5\rangle.x_2?(eq).\mathbf{0}$$

The system is given by

$$(\boldsymbol{\nu} x_1 x_2)\ \big(\text{server} \mid \text{client}\big)$$

Where

$$x_1 : \texttt{?Int.?Int.!Bool.end}$$

and

$$x_2 : \texttt{!Int.!Int.?Bool.end}$$

Session Types in Practice: Equality Test

$$\text{server} \stackrel{\text{def}}{=} x_1?(v).x_1?(w).x_1!\langle v == w\rangle.\mathbf{0}$$

$$\text{client} \stackrel{\text{def}}{=} x_2!\langle 3\rangle.x_2!\langle 5\rangle.x_2?(eq).\mathbf{0}$$

The system is given by

$$(\boldsymbol{\nu} x_1 x_2)\ \big(\text{server} \mid \text{client}\big)$$

Where

$$x_1 : \texttt{?Int.?Int.!Bool.end}$$

and

$$x_2 : \texttt{!Int.!Int.?Bool.end}$$

Research Timeline

$$\begin{tikzcd}[column sep=small]
 & \text{Milner, Parrow, Walker 1989/1992} \arrow[dl] \arrow[dr] & \\
\text{Honda 1993} \arrow[d] & & \text{Milner 1993} \arrow[d] \\
\text{Takeuchi, Honda, Kubo 1994} \arrow[d] & & \text{Pierce, Sangiorgi 1993} \arrow[d] \\
\text{Honda, Vasconcelos, Kubo 1998} \arrow[d] & & \text{Kobayashi, Pierce, Turner 1996} \arrow[d] \\
\text{To be continued...} & & \text{Sangiorgi 1998}
\end{tikzcd}$$

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. [PS96]

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. [PS96]
- $\ell_i T/\ell_o T$: 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. [PS96]
- $\ell_i T/\ell_o T$: channel used *only* in input/output and *exactly once* to transmit data of type T. [KPT96]
- $\langle l_j : T_j \rangle_{j \in J}$: labelled disjoint union of types. [Sangio97]

Key words for standard π- types

For session-typed π- calculus:

1. Sequentiality
2. Duality
3. Connection
4. Branch/Select

Key words for standard π- types

For session-typed π- calculus:

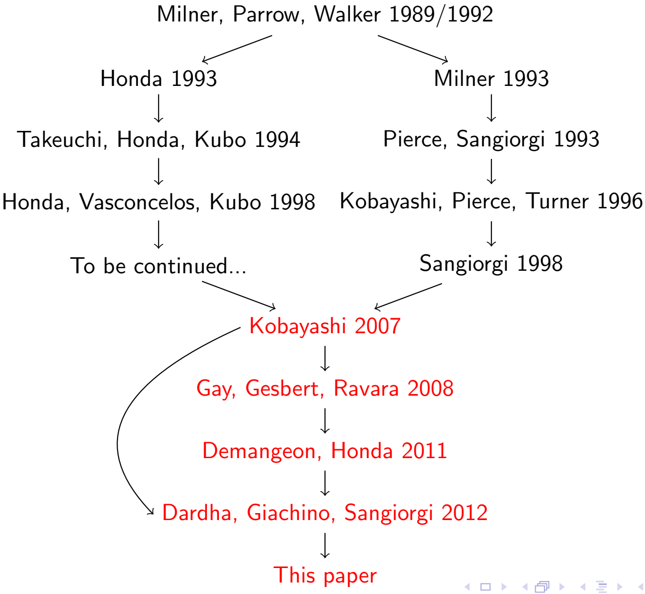
1. Sequentiality
2. Duality
3. Connection
4. Branch/Select

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



Research Timeline

Milner, Parrow, Walker 1989/1992

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

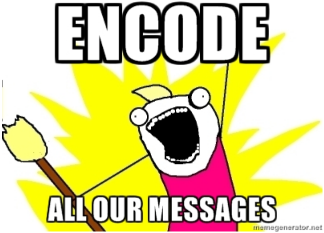
Milner 1993 → Pierce, Sangiorgi 1993 → Kobayashi, Pierce, Turner 1996 → Sangiorgi 1998

Kobayashi 2007

Dardha, Giachino, Sangiorgi 2012

This paper

Encoding session types into standard π-types



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.
- $\&\{l_j : S_j\}_{j \in J}$ and $\oplus\{l_j : S_j\}_{j \in J}$ are encoded as $\langle l_j : S_j \rangle_{j \in J}$
- Continuation of a session type becomes carried type: from breadth to depth.
- 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.

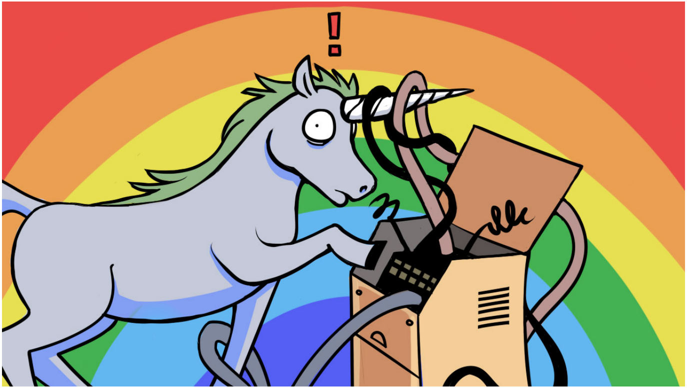
Motivation for this paper

- Limitation of "Session types revisited" [DGS12]: no infinite behaviours, no recursive session types.
- This work adds recursive session types and their encoding.
- Syntax of session types augmented with type variable X and recursive type construct $\mu X.S$.

Motivation for this paper

- Recursive session types and inductive duality $\bar{\cdot}$ don't go well together. (e.g $\mu X.!X$).
- $\bar{\cdot}$ does not commute with unfolding of recursive types.
- "Session types revisited", revisited: we use the complement function cplt() in the encoding. [BP12, BH13]

A bit more technical...



Session Types: Syntax

S	$::=$	$\mathtt{end}$	termination
		$!T.S$	send
		$?T.S$	receive
		$\oplus\{l_j : S_j\}_{j\in J}$	select
		$\&\{l_j : S_j\}_{j\in J}$	branch
T	$::=$	S	session type
		$\sharp T$	standard channel type
		$\mathtt{Bool}$	boolean type

Linear Types: Syntax

τ	$::=$	$\emptyset[]$	channel with no capability
		$\ell_i[\widetilde{T}]$	linear input
		$\ell_o[\widetilde{T}]$	linear output
		$\ell_\sharp[\widetilde{T}]$	linear connection
T	$::=$	τ	linear channel type
		$\langle l_j : T_j \rangle_{j \in J}$	variant type
		$\sharp T$	standard channel type
		`Bool`	boolean type

Encoding Session Types: Formally

$$
\begin{aligned}
[\![\mathtt{end}]\!] &\stackrel{\mathtt{def}}{=} \emptyset[] \\
[\![!T.S]\!] &\stackrel{\mathtt{def}}{=} \ell_o[[\![T]\!], [\![\overline{S}]\!]] \\
[\![?T.S]\!] &\stackrel{\mathtt{def}}{=} \ell_i[[\![T]\!], [\![S]\!]] \\
[\![\oplus\{l_j : S_j\}_{j\in J}]\!] &\stackrel{\mathtt{def}}{=} \ell_o[\langle l_j : [\![\overline{S_j}]\!]\rangle_{j\in J}] \\
[\![\&\{l_j : S_j\}_{j\in J}]\!] &\stackrel{\mathtt{def}}{=} \ell_i[\langle l_j : [\![S_j]\!]\rangle_{j\in J}]
\end{aligned}
$$

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

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

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

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

Then

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

Encoding Finite Session Types: Example

Let

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

Then

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

Encoding Finite Session Types: Example

Let

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

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:

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

and

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

Remark

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

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

Properties of the Encoding

Theorem (On types)

Encoding preserves typability of programs.

Theorem (On reductions)

Encoding preserves evaluation of programs.

Lemma (On duality relation)

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

Deriving properties from the encoding

Theorem (Subject Reduction)

Proof.
'On types', 'On reductions' and Subject Reduction in linearly-typed π- calculus. $\square$

Adding recursive session types...

Recursive Session Types: Syntax

S	$::=$	$\texttt{end}$	termination
		$!T.S$	send
		$?T.S$	receive
		$\oplus\{l_j : S_j\}_{j\in J}$	select
		$\&\{l_j : S_j\}_{j\in J}$	branch
		$X, \overline{X}$	type variable
		$\mu X.S$	recursive type
T	$::=$	S	session type
		$\sharp T$	standard channel type
		$\texttt{Bool}$	boolean type
		$X, \overline{X}$	type variable
		$\mu X.T$	recursive type

Recursive Linear Types: Syntax

τ	$::=$	$\emptyset[]$	channel with no capability
		$\ell_i[\widetilde{T}]$	linear input
		$\ell_o[\widetilde{T}]$	linear output
		$\ell_\sharp[\widetilde{T}]$	linear connection
T	$::=$	τ	linear channel type
		$\langle l_j : T_j \rangle_{j \in J}$	variant type
		$\sharp T$	standard channel type
		$\texttt{Bool}$	boolean type
		$X, \overline{X}$	type variable
		$\mu X.T$	recursive type

Encoding (Recursive) Session Types: Formally

$$
\begin{aligned}
[\![\mathtt{end}]\!] &\stackrel{\mathtt{def}}{=} \emptyset[] \\
[\![!T.S]\!] &\stackrel{\mathtt{def}}{=} \ell_o[[\![T]\!], [\![\overline{S}]\!]] \\
[\![?T.S]\!] &\stackrel{\mathtt{def}}{=} \ell_i[[\![T]\!], [\![S]\!]] \\
[\![\oplus\{l_j : S_j\}_{j\in J}]\!] &\stackrel{\mathtt{def}}{=} \ell_o[\langle l_j : [\![\overline{S_j}]\!]\rangle_{j\in J}] \\
[\![\&\{l_j : S_j\}_{j\in J}]\!] &\stackrel{\mathtt{def}}{=} \ell_i[\langle l_j : [\![S_j]\!]\rangle_{j\in J}] \\
[\![X]\!] &\stackrel{\mathtt{def}}{=} X \\
[\![\overline{X}]\!] &\stackrel{\mathtt{def}}{=} \overline{X} \\
[\![\mu X.S]\!] &\stackrel{\mathtt{def}}{=} \mu X.[\![S]\!]
\end{aligned}
$$

Encoding (Recursive) Session Types: Formally

$$
\begin{aligned}
[\![\mathsf{end}]\!] &\stackrel{\mathrm{def}}{=} \emptyset[] \\
[\![!T.S]\!] &\stackrel{\mathrm{def}}{=} \ell_o[[\![T]\!], [\![[\![\mathsf{cplt}(S)]\!]]\!]] \\
[\![?T.S]\!] &\stackrel{\mathrm{def}}{=} \ell_i[[\![T]\!], [\![S]\!]] \\
[\![\oplus\{l_j : S_j\}_{j \in J}]\!] &\stackrel{\mathrm{def}}{=} \ell_o[\langle l_j : [\![[\![\mathsf{cplt}(S_j)]\!]]\!]\rangle_{j \in J}] \\
[\![\&\{l_j : S_j\}_{j \in J}]\!] &\stackrel{\mathrm{def}}{=} \ell_i[\langle l_j : [\![S_j]\!]\rangle_{j \in J}] \\
[\![X]\!] &\stackrel{\mathrm{def}}{=} X \\
[\![\overline{X}]\!] &\stackrel{\mathrm{def}}{=} \overline{X} \\
[\![\mu X.S]\!] &\stackrel{\mathrm{def}}{=} \mu X.[\![S]\!]
\end{aligned}
$$

Properties of the extended encoding

- $\mathsf{cplt}(\cdot)$ and $\bar{\cdot}$ coincide for finite session types.
- This encoding is a conservative extension of the former.
- Faithfulness of the encoding still holds.
- Equality of carried types means type equality.
  Previously, syntactic identity.

Encoding Recursive Session Types: Example

$$
\begin{aligned}
\llbracket U \rrbracket &= \llbracket \mu X.\&\{l : X\} \rrbracket \\
&= \mu X.\llbracket \&\{l : X\} \rrbracket \\
&= \mu X.\ell_i[\langle l : \llbracket X \rrbracket \rangle] \\
&= \mu X.\ell_i[\langle l : X \rangle] = \upsilon
\end{aligned}
$$

$$
\begin{aligned}
\llbracket T \rrbracket &= \llbracket \mu X. \oplus \{l : X\} \rrbracket \\
&= \mu X.\llbracket \oplus\{l : X\} \rrbracket \\
&= \mu X.\ell_o[\langle l : \llbracket \mathrm{cplt}(X) \rrbracket \rangle] \\
&= \mu X.\ell_o[\langle l : \overline{X} \rangle] = \tau
\end{aligned}
$$

Let's discuss duality

- $\textsc{unf}(U) = \&\{l : \mu X.\&\{l : X\}\}$ dual of $\textsc{unf}(T) = \oplus\{l : \mu X. \oplus \{l : X\}\}$
- $\textsc{unf}(\upsilon) = \ell_i[\langle l : \upsilon\rangle] = \ell_i[\langle l : \mu X.\ell_i[\langle l : X\rangle]\rangle]$ dual of $\textsc{unf}(\tau) = \ell_o[\langle l : \mathsf{picplt}(\tau)\rangle] = \ell_o[\langle l : \mu X.\ell_i[\langle l : \mathsf{picplt}(\tau)\rangle]\rangle]$
- (because $\overline{X}\{\tau/X\} = \mathsf{picplt}(\tau)$ and $\mathsf{picplt}(\tau) = \mu X.\ell_i[\langle l : \mathsf{picplt}(\tau)\rangle]$)

Conclusions and Future Work 1/2

- Understanding the complexity and expressivity of (recursive) session types.
- Presented an encoding of (recursive) session types into (recursive) linear π- types.
- The encoding is a conservative extension of the former one on finite session types.
- There is also an encoding of processes!

Conclusions and Future Work 2/2

- The encoding allows derivation of basic properties (SR, TS...) of session π from standard typed π.
- The encoding allows elimination of redundancy in the syntax of session types and terms.
- Recursion and asynchrony.
- Case studies: polymorphism, higher-order.

Questions?

References I

G. Bernardi, O. Dardha, S. J. Gay, and D. Kouzapas.
On duality relations for session types.
To appear in Proc. of TGC, 2014.

G. Bernardi and M. Hennessy.
Using higher-order contracts to model session types.
CoRR, abs/1310.6176, 2013.

O. Dardha.
Recursive session types revisited, 2014.
Online extened version at http://www.dcs.gla.ac.uk/~ornela/my_papers/D14-Extended.pdf.

References II

O. Dardha.
Type Systems for Distributed Programs: Components and Sessions.
PhD thesis, University of Bologna, 2014.
http://www.dcs.gla.ac.uk/~ornela/my_papers/DardhaPhDThesis.pdf.

O. Dardha, E. Giachino, and D. Sangiorgi.
Session types revisited.
In *PPDP*, pages 139–150, New York, NY, USA, 2012. ACM.

Encoding Session Processes: Formally

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

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

Properties of the encoding

Theorem (Correctness of the Encoding)

$\Gamma \vdash P$ if and only if $[\![\Gamma]\!]_f \vdash [\![P]\!]_f$.

Theorem (Operational Correspondence)

Let P be a session process. The following hold.

1. *If $P \to P'$ then $[\![P]\!]_f \to\hookrightarrow [\![P']\!]_f$,*
2. *If $[\![P]\!]_f \to Q$ then, $\exists\ P', \mathcal{E}[\cdot]$ such that $\mathcal{E}[P] \to \mathcal{E}[P']$ and $Q \hookrightarrow [\![P']\!]_{f'}$, where f' is the updated f after reduction and $f_x = f_y$ for all $(\boldsymbol{\nu} xy) \in \mathcal{E}[\cdot]$.*

Operational Semantics of Session π-calculus

$$(\text{R-Com}) \qquad (\boldsymbol{\nu} xy)(x!\langle v\rangle.P \mid y?(z).Q) \to (\boldsymbol{\nu} xy)(P \mid Q[v/z])$$

$$(\text{R-Sel}) \quad (\boldsymbol{\nu} xy)(x \triangleleft l_j.P \mid y \triangleright \{l_i : P_i\}_{i\in I}) \to (\boldsymbol{\nu} xy)(P \mid P_j) \; j \in I$$

Encoding Session Processes: Formally

$$
\begin{array}{lcl}
[\![x!\langle v\rangle.P]\!]_f & = & (\boldsymbol{\nu} c)f_x!\langle v, c\rangle.[\![P]\!]_{f,\{x\mapsto c\}} \\
[\![x?(y).P]\!]_f & = & f_x?(y, c).[\![P]\!]_{f,\{x\mapsto c\}} \\
[\![x \triangleleft l_j.P]\!]_f & = & (\boldsymbol{\nu} c)f_x!\langle l_j\_c\rangle.[\![P]\!]_{f,\{x\mapsto c\}} \\
[\![x \triangleright \{l_i : P_i\}_{i\in I}]\!]_f & = & f_x?(y).\ \textbf{case}\ y\ \textbf{of}\ \{l_i\_c \triangleright [\![P_i]\!]_{f,\{x\mapsto c\}}\}_{i\in I} \\
[\![(\boldsymbol{\nu} xy)P]\!]_f & = & (\boldsymbol{\nu} c)[\![P]\!]_{f,\{x,y\mapsto c\}} \\
[\![P \mid Q]\!]_f & = & [\![P]\!]_f \mid [\![Q]\!]_f \\
[\![*P]\!]_f & = & *[\![P]\!]_f \\
[\![\mathbf{0}]\!]_f & = & \mathbf{0}
\end{array}
$$

Encoding Replication: Example

$$
\begin{aligned}
[\![P]\!]_f &= [\![*\big(a?(x).x \triangleleft l.a!\langle x\rangle.\mathbf{0}\big)]\!]_f \\
&= *[\![\big(a?(x).x \triangleleft l.a!\langle x\rangle.\mathbf{0}\big)]\!]_f \\
&= *\big(a?(x).[\![x \triangleleft l.a!\langle x\rangle.\mathbf{0}]\!]_f\big) \\
&= *\big(a?(x).(\boldsymbol{\nu} c)x!\langle l\_c\rangle.[\![a!\langle x\rangle.\mathbf{0}]\!]_{f,\{x\mapsto c\}}\big) \\
&= *\big(a?(x).(\boldsymbol{\nu} c)x!\langle l\_c\rangle.a!\langle c\rangle.\mathbf{0}\big)
\end{aligned}
$$

Encoding Replication: Example

$$
\begin{aligned}
[\![Q]\!]_f &= [\![*\big(b?(x).x \triangleright \{l : b!\langle x\rangle.\mathbf{0}\}\big)]\!]_f \\
&= *[\![\big(b?(x).x \triangleright \{l : b!\langle x\rangle.\mathbf{0}\}\big)]\!]_f \\
&= *\big(b?(x).[\![x \triangleright \{l : b!\langle x\rangle.\mathbf{0}\}\big)]\!]_f \\
&= *\big(b?(x).x?(y).\textbf{case}\; y \;\textbf{of}\; \{l\_c \triangleright [\![b!\langle x\rangle.\mathbf{0}\}]\!]_{f,\{x\mapsto c\}}\}\big) \\
&= *\big(b?(x).x?(y).\textbf{case}\; y \;\textbf{of}\; \{l\_c \triangleright b!\langle c\rangle.\mathbf{0}\}\}\big)
\end{aligned}
$$