on Polymorphic Sessions and Functions

a Tale of Two Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

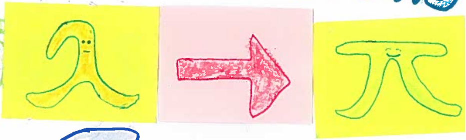
on Polymorphic Sessions and Functiλ Π

a Tale of Two Fully Abstract Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

on Polymorphic Sessions and Functi



a Tale of Two Fully Abstract Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

on Polymorphic Sessions and Functi

a Tale of Two Encodings

$\lambda \leftarrow \pi$

New

Fully Abstract

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

The π-calculus as a Descriptive Tool

by Kohei Honda
1995

$$\lambda \qquad M ::= x \mid \lambda x.M \mid MN$$

$$\pi \qquad P ::= \Sigma \pi_i.P_i \mid P|Q \mid (\nu x)P \mid !P \mid 0.$$

with $\pi ::= x(\tilde{y}) \mid \bar{x}\langle \tilde{y} \rangle.$

Milner's Encoding
1991

λ in π

$$[\![x]\!]_u \stackrel{\text{def}}{=} \bar{x}\langle u \rangle.$$

$$[\![\lambda x.M]\!]_u \stackrel{\text{def}}{=} u(xu').[\![M]\!]_{u'}$$

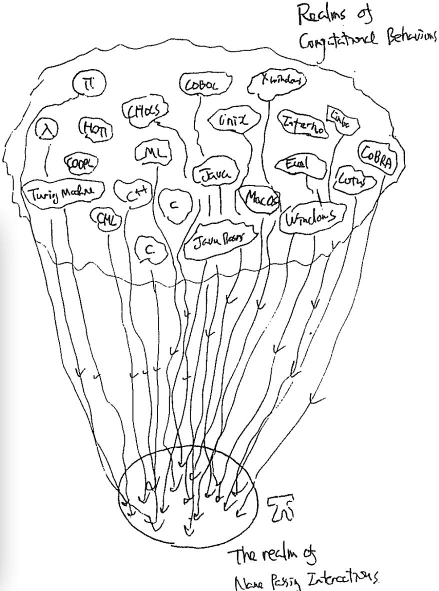
$$[\![MN]\!]_u \stackrel{\text{def}}{=} (\nu f x)\left([\![M]\!]_f \mid \bar{f}\langle xu \rangle \mid [\![x = N]\!]\right)$$

with $[\![x = N]\!] \stackrel{\text{def}}{=} !x(u').[\![N]\!]_{u'}.$

* Examples of Representable Computation.

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [HY94]
- strategies on Games [HO95]

⋮

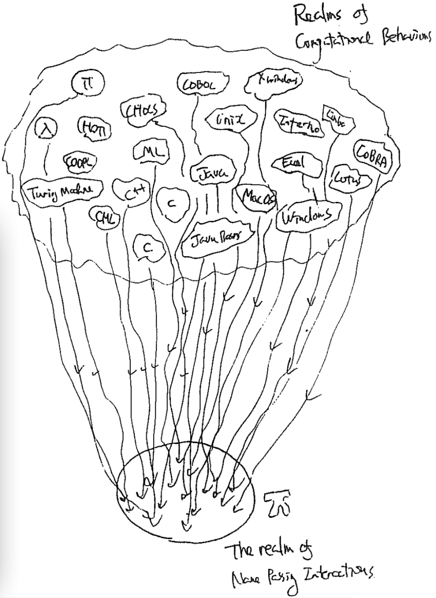


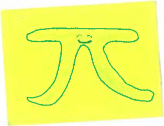
* Examples of Representable Computation.

Operationally Sound but NOT Fully Abstract

- λ-calculus [MPW89, Milner90, Milner92, …]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, …]
- Proof Nets [Bellin and Scott 93]
- Arbitrary "constant" interaction [HY94]
- strategies on Games [HO95]

⋮

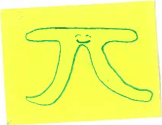




$$M \simeq N \;\Longleftarrow\; [\![M]\!] \simeq [\![N]\!]$$

Conextual Congruence

Contextual Congruence



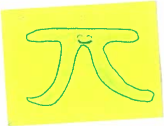
$$M \simeq N \quad \Longleftarrow \quad [\![M]\!] \simeq [\![N]\!]$$

Conextual congruence

Contextual Congruence

$$C[P]\Downarrow_a \text{ iff } C[Q]\Downarrow_a$$

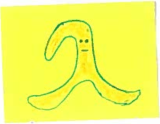
$$C[[\![M]\!]]\Downarrow_a \text{ iff } C[[\![N]\!]]\Downarrow_a$$

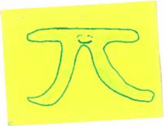


$$M \simeq N \;\underset{\text{NOT}}{\not\Rightarrow}\; [\![M]\!] \simeq [\![N]\!]$$

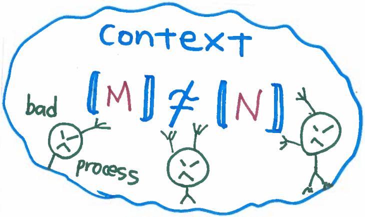
$$C[P]\Downarrow_a \text{ iff } C[Q]\Downarrow_a$$

$$C[[\![M]\!]]\Downarrow_a \text{ iff } C[[\![N]\!]]\Downarrow_a$$



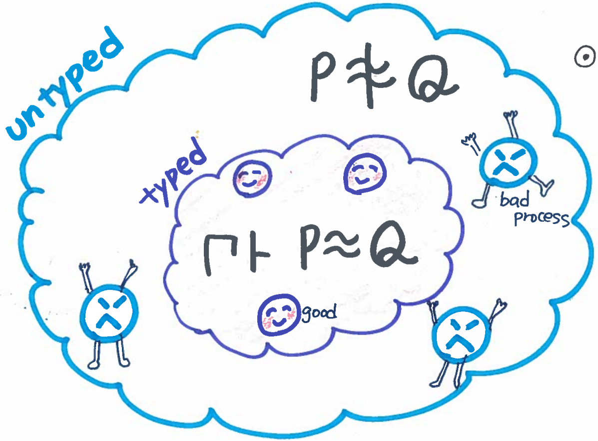


$$M \simeq N \quad \underset{\text{NOT}}{\not\Rightarrow} \quad [\![M]\!] \simeq [\![N]\!]$$



Typed Semantics in π 1991 →

IO-subtyping, Linear types, Secure Information Flow, ...



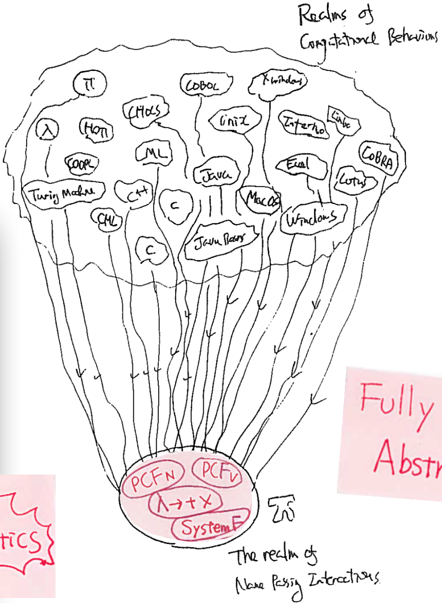
- Correctness of Encoding
- Limit environment Γ
  $\Rightarrow$ Equate <u>more</u> processes
- Compositional

* Examples of Representable Computation.

- λ-calculus [MPW89, Milner90, Milner92, …]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, …]
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [HY94]
- strategies on Games [HO95]

⋮

Game Semantics



Fully Abstract

* Examples of Representable Computation.

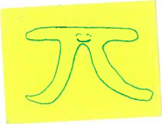
- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 9
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [C
- strategies on Games [HO95]

⋮

Complicated ...

Game Semantics

Fully Abstract



$$M \simeq N \Leftarrow [\![M]\!] \simeq [\![N]\!]$$

λ System F

π Session ILL

$$M \simeq N \;\leftarrow\; [\![M]\!] \simeq [\![N]\!]$$

λ System F

π Session ILL

$$M \simeq N \quad \Longrightarrow \quad [\![M]\!] \simeq [\![N]\!]$$

λ System F

π Session ILL

$M \simeq N$

$[\![M]\!] \simeq [\![N]\!]$

$[\![P]\!] \simeq [\![Q]\!] \Rightarrow P \simeq Q$

Reverse

New

λ System F

π Session ILL

$$M \simeq N \qquad [\![M]\!] \simeq [\![N]\!]$$

$$[\![P]\!] \simeq [\![Q]\!] \longleftarrow P \simeq Q$$

Reverse

New

λ System F

π Session ILL

$M \simeq N$ $\qquad$ $[\![M]\!] \simeq [\![N]\!]$

$[\![P]\!] \simeq [\![Q]\!] \leftarrow P \simeq Q$

TWO APPLICATIONS

Session π Caires, Pérez, Pfenning, Toninho 13

Types

$$A ::= A \otimes B \mid A \multimap B \mid \mathbf{1} \mid\ !A$$
$$\mid \forall X.A \mid \exists X.A \mid X$$

Processes

$$P ::= x\langle y\rangle.P \mid x(y).P \mid 0 \mid\ !x(y).P$$
$$\mid x(Y).P \mid x\langle B\rangle.P \mid [x \leftrightarrow y]$$
$$\mid (\nu x)P \mid (P \mid Q)$$

Judgement

Process

$$\underbrace{X_1, \ldots, X_k}_{\text{Poly Vars}} ; a_1 : A_1, \ldots, a_n : A_n \vdash P :: b : A$$

name Type

name type

process P provides A along b if composed with sessions $\vec{a} : \vec{A}$

Judgement

Process

$$\underbrace{X_1, \dots, X_k}_{\text{Poly Vars}} ; a_1 : A_1, \dots, a_n : A_n \vdash P :: b : A$$

name Type name type

process P provides A along b if composed with sessions $\vec{a} : \vec{A}$

Judgement

Process

$$\underbrace{X_1, \ldots, X_k}_{\text{Poly Vars}} ; a_1 : A_1, \ldots, a_n : A_n \vdash P :: b : A$$

name Type name type

Cut Elmination

$$\frac{\Delta_1 \vdash P_1 :: a : A \qquad \Delta_2, a : A \vdash P_2 :: b : B}{\Delta_1, \Delta_2 \vdash (\nu a)(P_1 \mid P_2) :: b : B}$$

Identity

$$\frac{}{a : A \vdash [a \leftrightarrow b] :: b : A}$$

Polymorphic Session

$$\forall R \quad \frac{X \, ; \Delta \vdash P :: a : A}{\Delta \vdash a(X).P :: a : \forall X. A} \qquad \text{Input}$$

$$\exists R \quad \frac{\Delta \vdash P :: a : A\{B/X\}}{\Delta \vdash a\langle B \rangle . P :: a : \exists X. A} \qquad \text{output}$$

Left rules define how to use a session of a given type

Polymorphic Session

Deadlock Free

Live

Strong Normalising

$$\forall R \quad \frac{X ; \Delta \vdash P :: a : A}{\Delta \vdash a(X).P :: a : \forall X.A}$$

Input

$\simeq_\pi$

$$\exists R \quad \frac{\Delta \vdash P :: a : A\{B/X\}}{\Delta \vdash a\langle B\rangle.P :: a : \exists X.A}$$

output

Barbed Congruence

Left rules define how to use a session of a given type

Linear F
Zhao, Zhang, Zdancewic 2010 

Types

$$A ::= A \otimes B \mid A \multimap B \mid {!A} \mid 1 \mid 2$$
$$\mid \forall X. A \mid \exists X. A \mid X$$

Terms

$$M, N ::= \lambda x. M \mid M N \mid \langle M \otimes N \rangle \mid \text{let } x \otimes y = M \text{ in } N$$
$$\mid {!M} \mid \text{let } {!u} = M \text{ in } N$$
$$\mid \Lambda X. M \mid M[A] \mid \text{pack } A \text{ with } M \mid \text{let } (X, y) = M \text{ in } N$$
$$\mid \text{let } 1 = M \text{ in } N \mid \langle \rangle \mid \mathbf{T} \mid \mathbf{F}$$

Encoding: from λ to π

Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

Intro $\Rightarrow$ Right / Elim $\Rightarrow$ Left + Cut + Identity

Encoding is FUN

Encoding: from λ to π

Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

Intro ⇒ Right / Elim ⇒ Left + Cut + Identity

$$[\![M]\!]_a$$

name

$$[\![x]\!]_a = [x \leftrightarrow a]$$

$$[\![\langle\rangle]\!]_a = 0$$

$$[\![\lambda x.M]\!]_a = a(x).[\![M]\!]_a$$

$$[\![MN]\!]_a = [\![M]\!]_x \mid \overline{x}(y).[\![N]\!]_y \mid [x \leftrightarrow a]$$

The π-calculus as a Descriptive Tool

λ in π

Milner's Encoding 1991

$$[\![x]\!]_u \stackrel{\text{def}}{=} \overline{x}\langle u\rangle.$$

$$[\![\lambda x.M]\!]_u \stackrel{\text{def}}{=} u(x u').[\![M]\!]_{u'}$$

$$[\![MN]\!]_u \stackrel{\text{def}}{=} (\nu f x)\left([\![M]\!]_f \mid \overline{f}\langle x u\rangle \mid [\![x = N]\!]\right)$$

with $[\![x = N]\!] \stackrel{\text{def}}{=} !x(u').[\![N]\!]_{u'}.$

Encoding: from λ to π

Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

Intro $\Rightarrow$ Right / Elim $\Rightarrow$ Left + Cut + Identity

λ in π

$$[\![x]\!]_u \stackrel{\text{def}}{=} \overline{x}\langle u\rangle.$$

$$[\![\lambda x.M]\!]_u \stackrel{\text{def}}{=} u(x u').[\![M]\!]u'$$

$$[\![M N]\!]_u \stackrel{\text{def}}{=} (\nu f x)\left([\![M]\!]_f \mid \overline{f}\langle x u\rangle \mid [\![x{=}N]\!]\right)$$

with $[\![x{=}N]\!] \stackrel{\text{def}}{=} !x(u').[\![N]\!]u'.$

$$[\![\,M\,]\!]_a$$

a ↑ name

$$[\![x]\!]_a = [x \leftrightarrow a]$$

$$[\![\langle\rangle]\!]_a = 0$$

$$[\![\lambda x.M]\!]_a = a(x).[\![M]\!]_a$$

$$[\![MN]\!]_a = [\![M]\!]_x \mid \overline{x}(y).[\![N]\!]_y \mid [x \leftrightarrow a]$$

From  to 

From Sequent Calculus to Natural Deduction

$$[\![P]\!]_{\Delta \vdash a : A} \quad \text{with} \quad \Delta \vdash P :: a : A$$

$$[\![0]\!] = \langle \rangle$$

$$[\![[x \leftrightarrow a]]\!] = x$$

$$[\![a(x).P]\!] = \lambda x.[\![P]\!]$$

$$[\![a(X).P]\!] = \Lambda X.[\![P]\!]$$

Parallel

$$\left[\!\left[\frac{\Delta_1 \vdash P :: a : A \qquad \Delta_2, a : A \vdash Q :: b : C}{\Delta_1, \Delta_2 \vdash (\nu a)(P \mid Q) :: b : C} \right]\!\right]$$

Substitute P into a in Q

$$= \frac{\Delta_2, a : A \vdash [\![Q]\!]_b : C \qquad \Delta_1 \vdash [\![P]\!]_a : A}{\Delta_1, \Delta_2 \vdash [\![Q]\!]\{[\![P]\!]/a\} : C}$$

Poly

$$[\![\, \underbrace{x\langle \overset{\text{Type}}{B} \rangle}_{\text{Application of } B \text{ to } x} . P \,]\!] = [\![P]\!]\{x[B]/x\}$$

replace x by $x[B]$

cf. $[\![\, x\langle b\rangle.(P \mid Q) \,]\!] = [\![Q]\!]\{(x[\![P]\!])/x\}$

Application of P to x

Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\!| [\![M]\!]_z |\!) \cong M$$

$$[\![(\!| P |\!)]\!]_z \cong P$$

Full Abstraction

$$[\![M]\!]_z \cong [\![N]\!]_z \quad \text{iff} \quad M \cong N$$

$$(\!| P |\!) \cong (\!| Q |\!) \quad \text{iff} \quad P \cong Q$$

Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\!| [\![M]\!]_z |\!) \cong M$$

$$[\![(\!| P |\!)]\!]_z \cong P$$

Definability

$$\forall P\ \exists M\quad [\![M]\!]_z \cong P$$

$$\forall M\ \exists P\quad (\!| P |\!) \cong M$$

Full Abstraction

$$[\![M]\!]_z \cong [\![N]\!]_z \quad \text{iff} \quad M \cong N$$

$$(\!| P |\!) \cong (\!| Q |\!) \quad \text{iff} \quad P \cong Q$$

Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\!|\, [\![M]\!]_z \,|\!) \cong M$$

$$[\![\, (\!| P |\!) \,]\!]_z \cong P$$

Full Abstraction

$$[\![M]\!]_z \cong [\![N]\!]_z \quad \text{iff} \quad M \cong N$$

$$(\!| P |\!) \cong (\!| Q |\!) \quad \text{iff} \quad P \cong Q$$

Derived

Theorems

Operational Correspondence / Type Preserving

Inverse

$$\llparenthesis \llbracket M \rrbracket_Z \rrparenthesis \cong M \quad ✓$$

$$\llbracket \llparenthesis P \rrparenthesis \rrbracket_Z \cong P \quad ✓$$

Full Abstraction

$$\llbracket M \rrbracket_Z \cong \llbracket N \rrbracket_Z \Rightarrow M \cong N$$

$$\llparenthesis P \rrparenthesis \cong \llparenthesis Q \rrparenthesis \Rightarrow P \cong Q$$

Application 1 Inductive and Coinductive Types

Parametric Poly is Expressive Enough to Encode Inductive / Co-Inductive Types

as Initial / Final (Co)Algebra

$$
\begin{tikzcd}
F(T_i) \arrow[r, "F(h)"] \arrow[d, "f"'] & F(X') \arrow[d, "f'"] \\
T_i \arrow[r, "h"'] & X'
\end{tikzcd}
$$

h unique

$$
\begin{tikzcd}
1+\mathrm{Nat} \arrow[r, "F(\mathrm{fold}[X'](f'))"] \arrow[d, "{[z, succ]}"'] & F(X') \arrow[d, "f'"] \\
\mathrm{Nat} \arrow[r, "\mathrm{fold}[X'](f')"'] & X'
\end{tikzcd}
$$

can be defined in System F and fold is unique

for all inductive types

Inductive

Coinductive Types

$$\begin{tikzcd} X' \arrow[r, "h"] \arrow[d, "f'"'] & T_f \arrow[d, "f"] \\ F(X') \arrow[r, "F(h)"'] & F(T_f) \end{tikzcd} \qquad \begin{tikzcd} X' \arrow[r, "\mathrm{unfold}[X](f')"] \arrow[d, "f'"] & \mathrm{NatStream} \arrow[d, "{[hd, tl]}"'] \\ F(X') \arrow[r, "F(\mathrm{unfold}[X](f'))"'] & \mathrm{Nat} \times \mathrm{NatStream} \end{tikzcd}$$

Question Sess Poly π is Expressive Enough?

Theorems

$\forall Q$ s.t. $u : F(X) \multimap X,\ y_1 : T_i \vdash Q :: y_2 : X$. $\mathrm{Fold}(X) \cong Q$

$\forall_A Q$ s.t. $u : A \multimap F(A),\ y_1 : A \vdash Q :: y_2 : T$, $Q \cong \mathrm{Unfold}(A)$

Application (2) HOπ + PRosess Monad

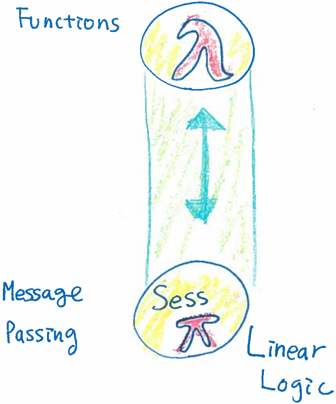
- Full Abstraction / Inverse $x\langle M\rangle.\,P$
- Strong Normalisation via Strong Normalisation

HOπ λ

$$P \to Q \quad \Rightarrow \quad (\!| P |\!) \xrightarrow{+} (\!| Q |\!)$$

cf. [CPPT'13] logical relation

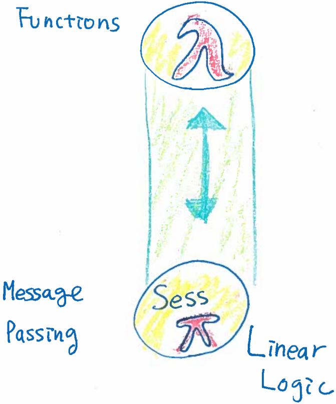
Summary



- Use Sess π (LL-based) to articulate boarder computations
- Algebraic Programming (F-algebra)

Summary

SAD?



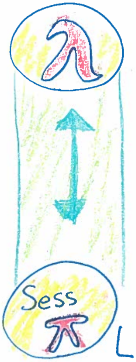
- Use LL-based Sess π to articulate boarder computations
- Algebraic Programming (F-algebra)

Summary

SAD?

NO!

Functions



Message Passing

Linear Logic

- Use LL-based Sess π to articulate boarder computations
- Algebraic Programming (F-algebra)