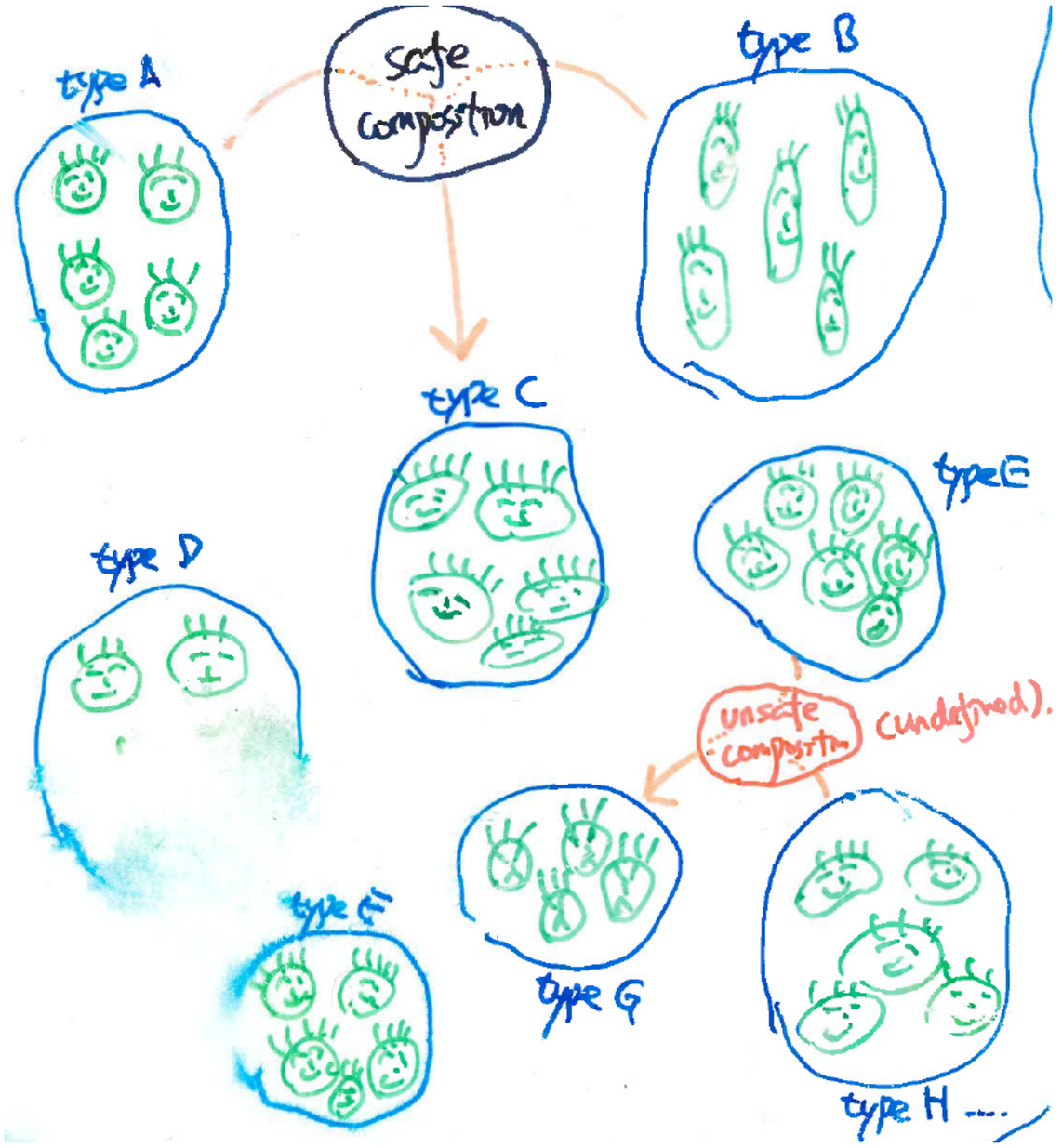
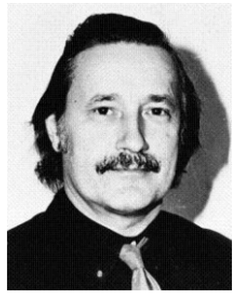
Linear Logic-Based Session Type: Expressiveness



Southern and Midlands Logic Seminar: 13 December 2023

Nonbuko Yoshida, University of Oxford

Christopher Strachey



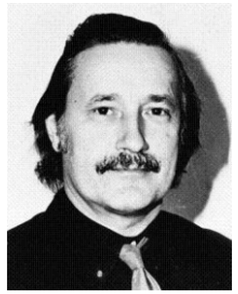
1916-1975

Computer Games, Literature and Music

A schoolmaster at Harrow

Fundamental Concepts of
Programming Languages

Christopher Strachey



1916-1975

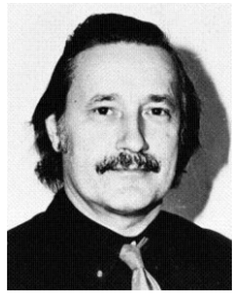
Computer Games, Literature and Music

A schoolmaster at Harrow

- **Christopher Strachey** (sequential computation)

Fundamental Concepts of Programming Languages

Christopher Strachey



1916-1975

Computer Games, Literature and Music

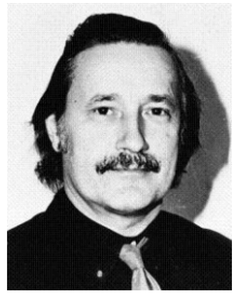
A schoolmaster at Harrow

- **Christopher Strachey** (sequential computation)

  Fundamental Concepts of Programming Languages

  - ***Types*** = **mathematical objects** and **abstract computation (data types, polymorphism)**

Christopher Strachey



1916-1975

Computer Games, Literature and Music

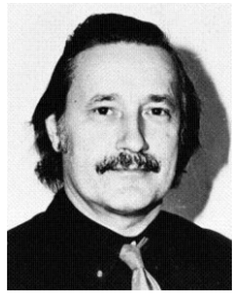
A schoolmaster at Harrow

- **Christopher Strachey** (sequential computation)

  Fundamental Concepts of Programming Languages

  - ***Types*** = **mathematical objects** and **abstract computation (data types, polymorphism)**
  - **Structured programming = High-level programming**

Christopher Strachey



1916-1975

Computer Games, Literature and Music

A schoolmaster at Harrow

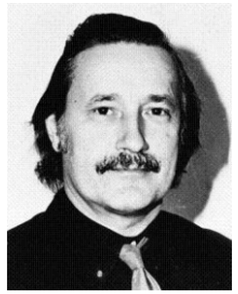
- **Christopher Strachey** (sequential computation)

  Fundamental Concepts of Programming Languages

  - ***Types*** = **mathematical objects** and **abstract computation (data types, polymorphism)**
  - **Structured programming = High-level programming**

- **Session types** (concurrency & communication)

Christopher Strachey



1916-1975

Computer Games, Literature and Music

A schoolmaster at Harrow

- **Christopher Strachey** (sequential computation)

  Fundamental Concepts of Programming Languages

  - ***Types*** = **mathematical objects** and **abstract computation (data types, polymorphism)**
  - **Structured programming = High-level programming**
- **Session types** (concurrency & communication)
  - Structured programming = ***protocols***

ETAPS 2019 TEST-OF-TIME AWARD

Language primitives and type discipline for structured communication-based programming' by Kohei Honda, Vasco T. Vasconcelos and Makoto Kubo

POPL 2008 MOST INFLUENTIAL PAPER AWARD

LICS 2018 TEST OF TIME AWARD

LICS'98

A fully abstract game semantics for general references

Samson Abramsky Kohei Honda
LFCS, University of Edinburgh
{samson,kohei}@dcs.ed.ac.uk

Guy McCusker
St John's College, Oxford
mccusker@comlab.ox.ac.uk

Abstract

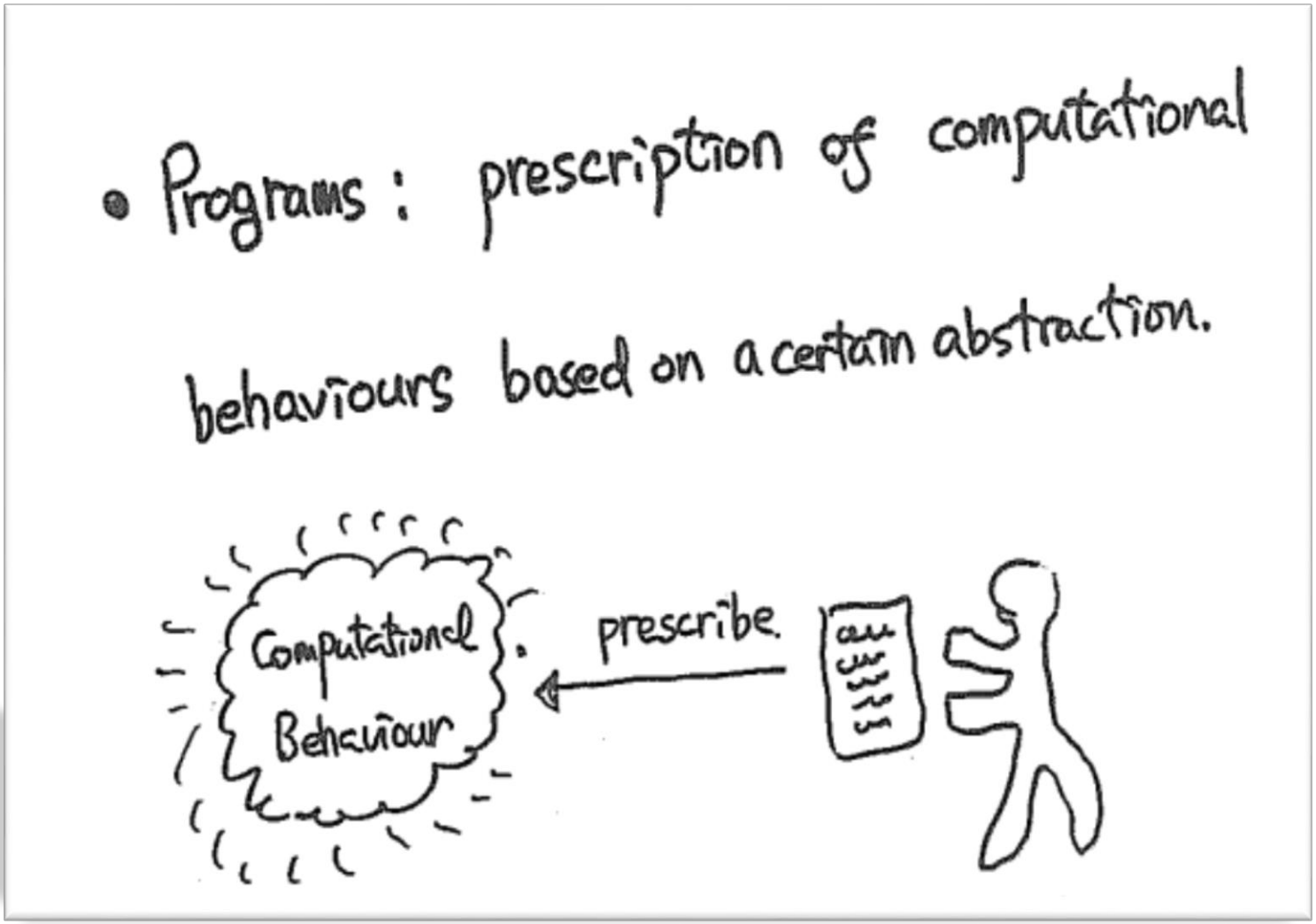
A games model of a programming language with higher-order store in the style of ML-references is introduced. The category used for the model is obtained by relaxing certain behavioural conditions on a category of games previously used to provide fully abstract models of pure functional languages. The model is shown to be fully abstract by means of factorization arguments which reduce the question of definability for the language with higher-order store to that for its purely functional fragment.

Idioms for Interaction

—Invitation to Hacking on π-calculus—

Programming languages are tools which offer frameworks of abstraction for such activities – promoting or limiting them

- *Imperative*
- *Functional*
- *Logical*



On Programs and Programming

- The most fundamental element of a PL

  in this context is a set of operations

  it is based on:

  Imperative: assignment, jump.

  Functional: β-reduction.

  Logical: unification.

- Another element is how we can combine,

  or structure, these operations:

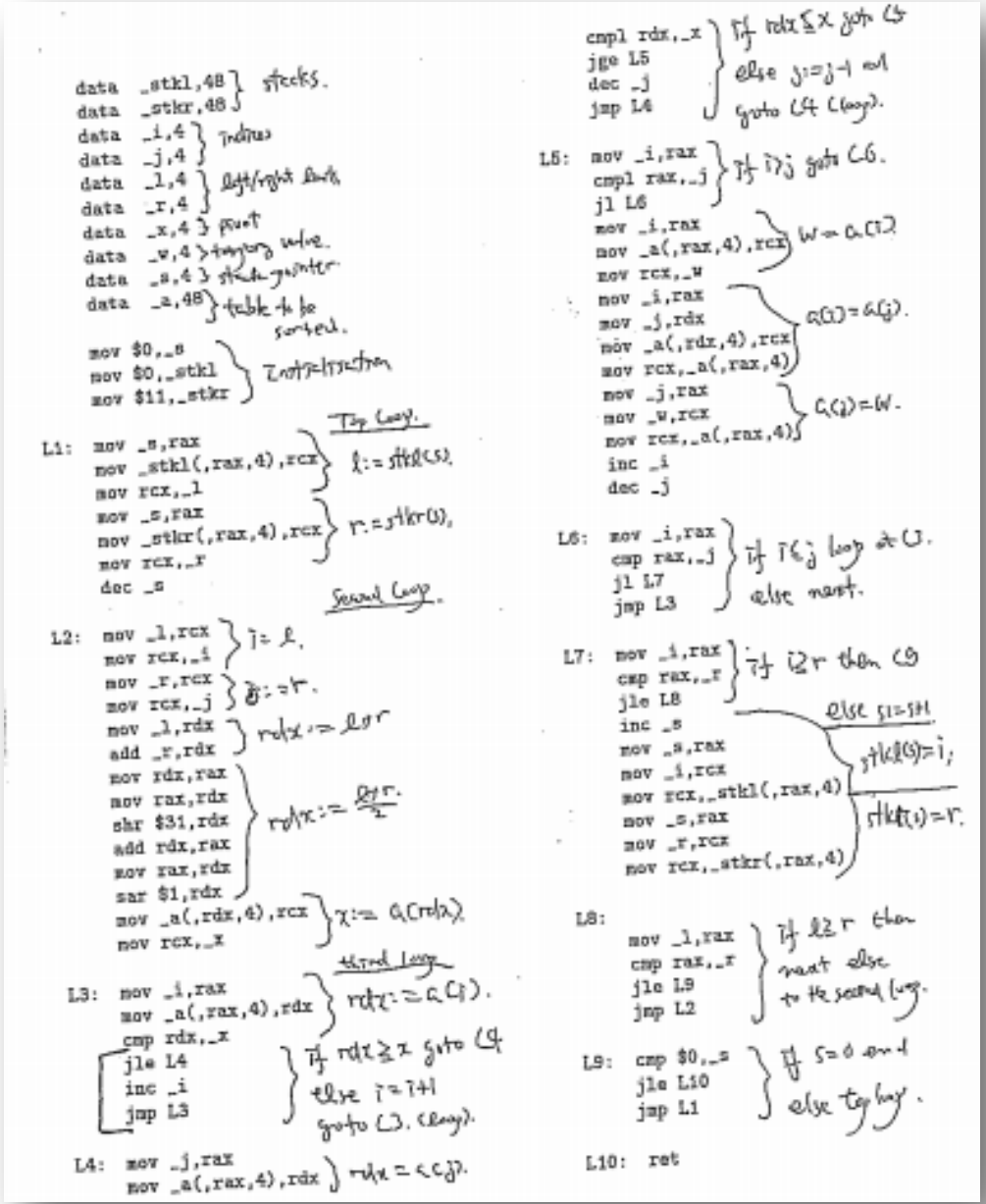
  Imperative: sequential composition, if-then-else,

  while, procedures, modules, ...

  Functional: application, product, union,

  recursion, modules, ...

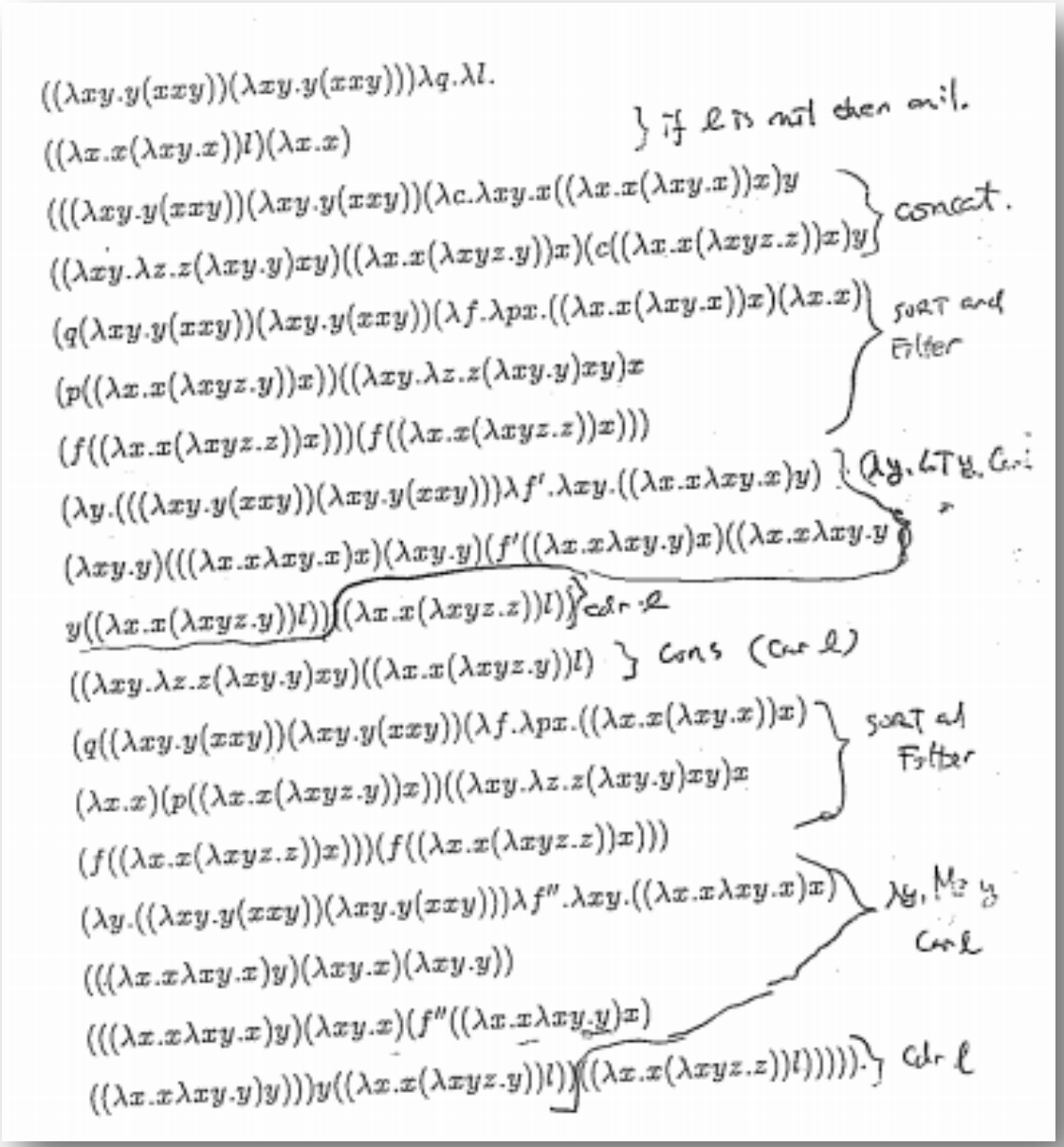
UNSTRUCTURED:



STRUCTURED:

```
Var a: array[MAX] of int;

Procedure sort(l,r: int);
  Var i,j, x: int;
  i:= l; j:= r;
  x:= [(l+r) div 2];          • Choose a pivot.
  repeat
    while a[i]<x do i:= i+1 end
    while a[j]>x do j:= j+1 end.     } Partition into two parts.
    if i≤j then swap(i,j); i:=i+1; j:=j-1; end
  until i>j;
  if l<j then sort(l,j);      } Recursively
  if l<r then sort(i,r);      } sort two parts.
end

Procedure swap(i,j: int)
    var w: int;
        w:= a[i]; a[i]:= a[j]; a[j]:= w;
end
```

Quicksort in pure lambda:



Quicksort with combinators:

$$Y(\lambda f.\ \lambda l.\ (\text{Isnil } l)\ (\text{Concat } (f\ \text{Filter } (\lambda y.\ LT\ y\ (\text{Car } l))\ (\text{Cdr } l))\ (\text{Cons } (\text{Car } l)\ (f\ \text{Filter } (\lambda y.\ \text{Me } y\ (\text{Car } l)\ (\text{Cdr } l))))))$$

$I = \lambda x.x \quad T = \lambda xy.x \quad F = \lambda xy.y \quad Y = (\lambda xy.y(xxy))(\lambda xy.y(xxy))$

$\text{Cons} = \lambda xy.\lambda z.zFxy \quad \text{Isnil} = \lambda x.xT$

$\text{Car} = \lambda x.x(\lambda xyz.y) \quad \text{Cdr} = \lambda x.x(\lambda xyz.z)$

$\text{Concat} = Y(\lambda c.\lambda xy.x\ (\text{Isnil } x)\ y\ (\text{Cons } (\text{Car } x)\ (c\ (\text{Cdr } x)\ y)))$

$\text{Filter} = Y(\lambda f.\lambda px.(\text{Isnil } x)\ I\ (p\ (\text{Car } x))\ (\text{Cons } x\ (\text{Filter Cdr } x))\ (\text{Filter } (\text{Cdr } x)))$

$\text{Iszero} = \lambda x.xT \quad \text{Pred} = \lambda x.xF$

$LT = Y(\lambda f.\lambda xy.(\text{Iszero } y)\ F\ ((\text{Iszero } x)\ T\ (f\ (\text{Pred } x)\ (\text{Pred } y))))$

$ME = Y(\lambda f.\lambda xy.(\text{Iszero } x)\ ((\text{Iszero } y)\ I\ F)\ ((\text{Iszero } y)\ (T)\ (f\ (\text{Pred } x)\ (\text{Pred } y))))$

Quicksort in ML:

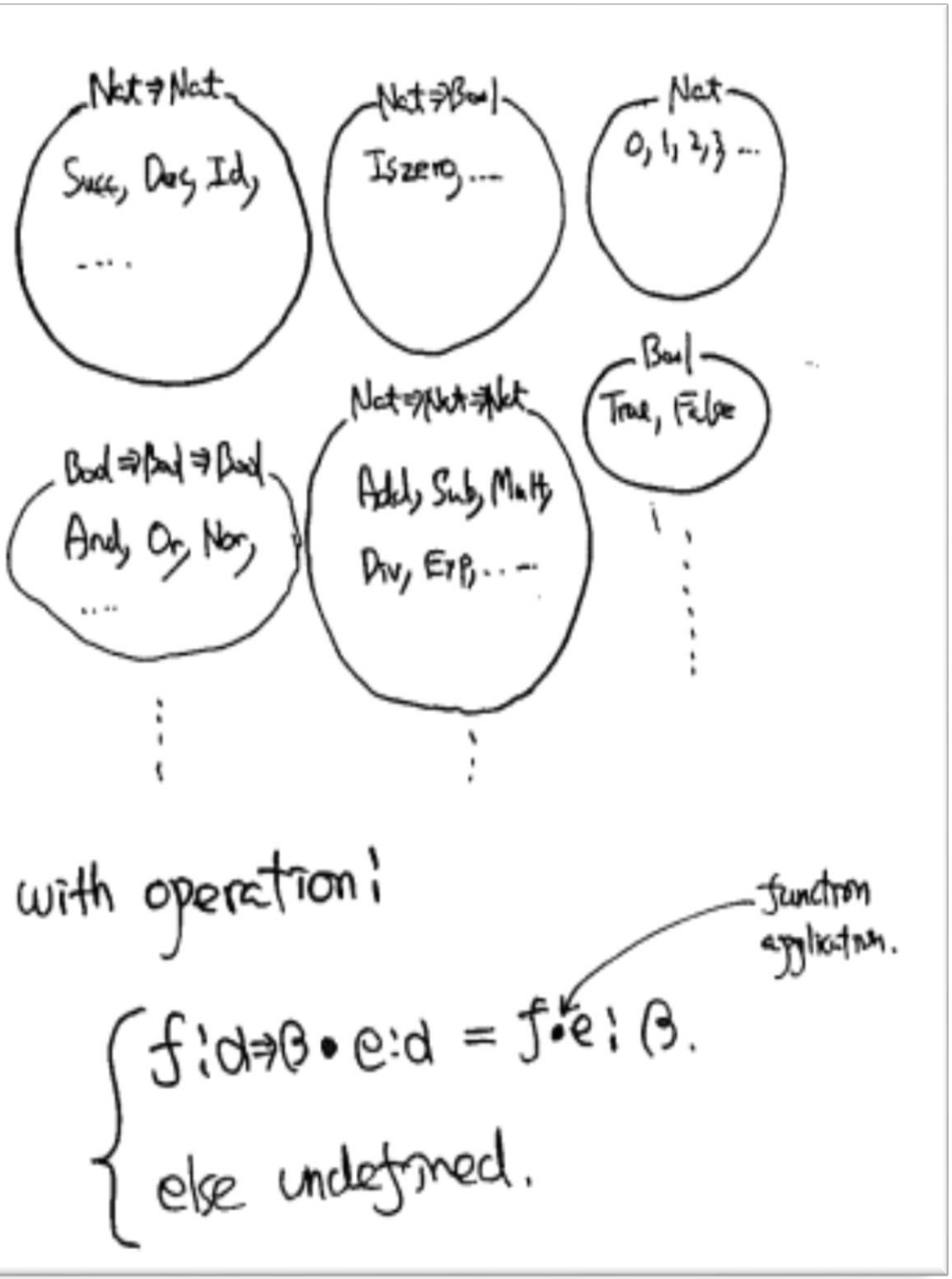
```
fun qs nil:int list = nil
  | qs (x::r) = let val small =
                          filter (fn y => y<x) r
                    and large =
                          filter (fn y => y>=x) r
                in qs small @ [x] @ qs large
                end

fun filter p nil = nil
  | filter p (x::r) =
      if p x then x:: filter p r
             else filter p r
```

The Role of Types in π-calculus

- (classification) How can we classify name-passing interactive behaviours, i.e. behaviours representable in π-calculus? What classes ("types") of behaviours can we find in the calculus?

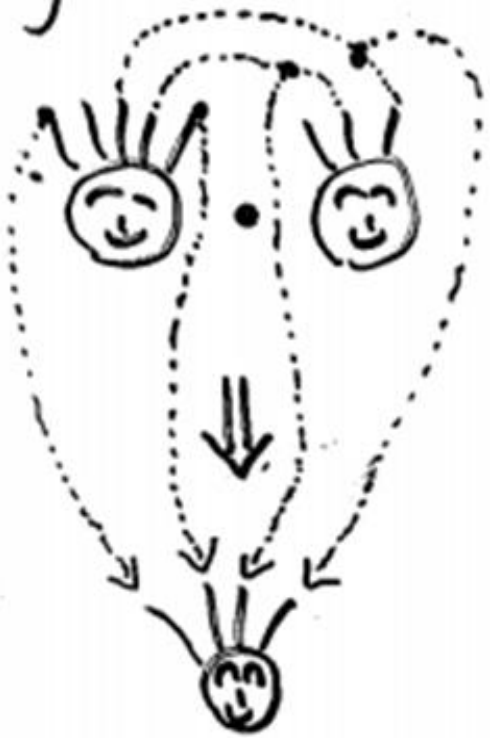
- (safety) Is this program/system in the safe (or correct, relevant, ...) classes of behaviours? Can the safety be preserved compositionally?

Functional Types



Process Types

- When it comes to processes, composition becomes:



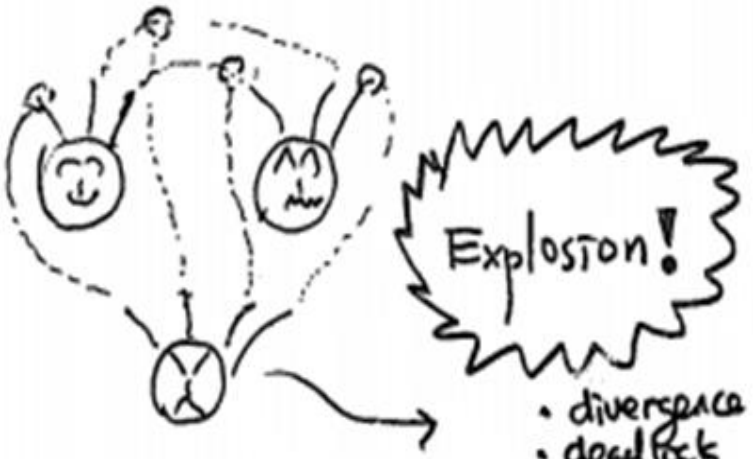
cf.

3 + 5

8

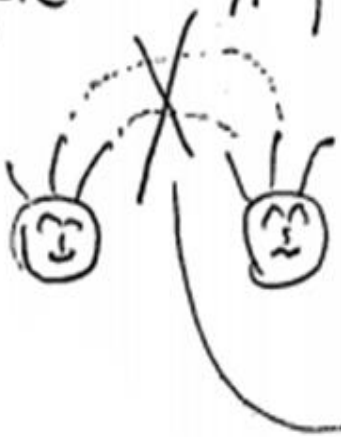
$P \bullet_\delta \delta \longmapsto_\tau \delta$

- But some composition is dangerous!



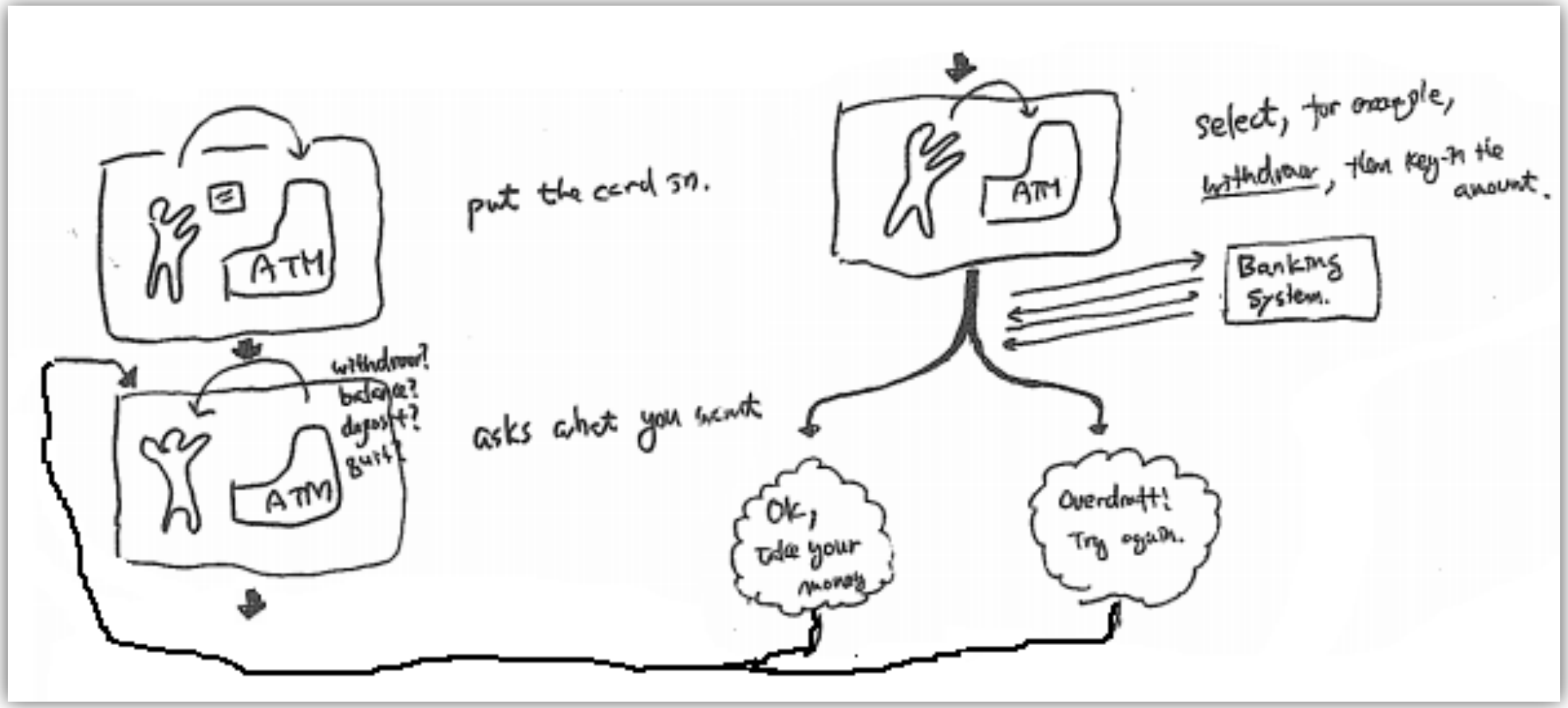
- divergence
- deadlock
- run-time error
- ⋮

- Therefore we type processes,



The connection is prohibited

Implementing ATM

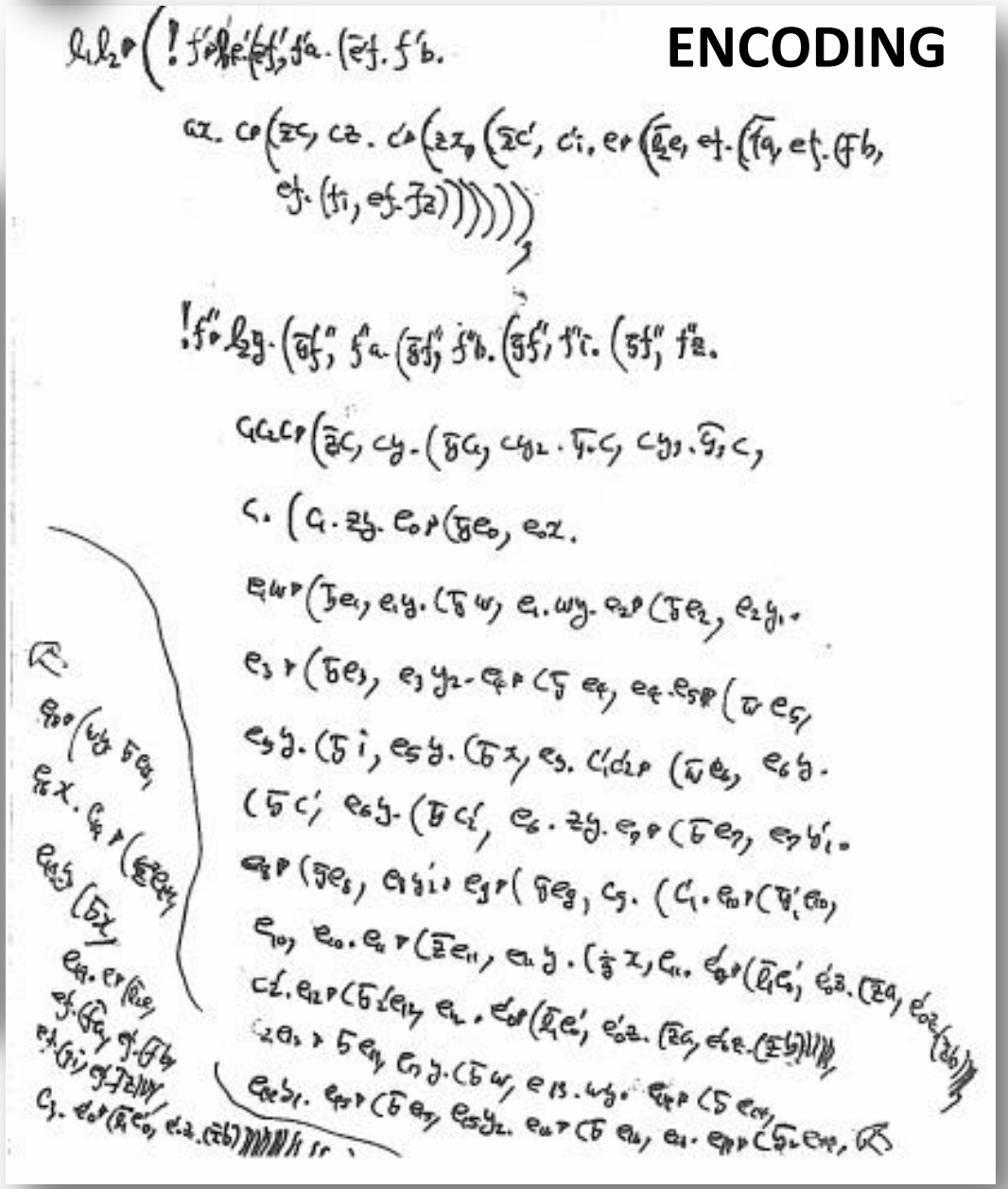


Implementing ATM

$ATM(cb) \stackrel{def}{=}$

```
ATM<z>:: z?ID; [?wd; ?X;                          * ↔user.
        b<w>:: w!wt; !ID; !X;                     * ↔bank.
          [?ok:                                   *  :
            z!ok; !X; ATM(cb),                    * ↔user
           ?overdraft:                            * ↔bank
            z!over; ATM(cb)],                     * ↔user.
        ?bd;                                      * ↔user
          b<w>:: w!bd; ?X;                        * ↔bank
            z!X; ATM<cb>]
```

ENCODING

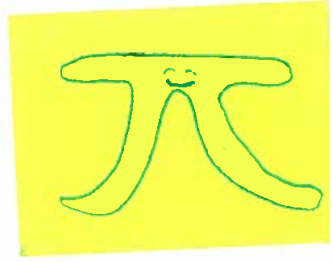


on Polymorphic Sessions and Functions or a Tale of Two Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

On Polymorphic Sessions and Functions

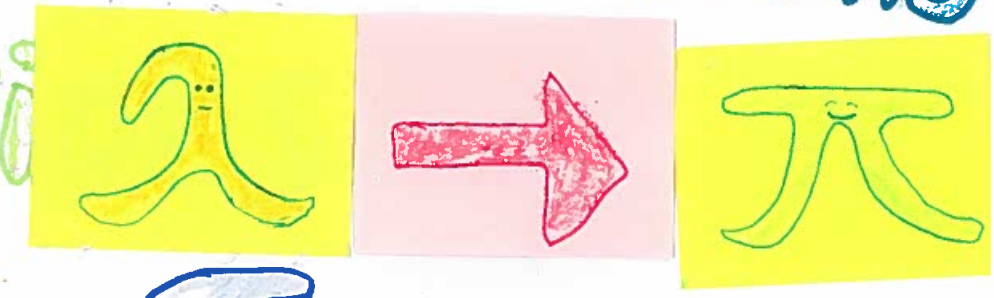


A Tale of Two Fully Abstract Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

On Polymorphic Sessions and Functions



A Tale of Two Fully Abstract Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

On Polymorphic Sessions and Functions

A Tale of Two Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

The π-calculus as a Descriptive Tool

by Kohei Honda 1995

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

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

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

Milner's Encoding 1991

λ in π

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

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

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

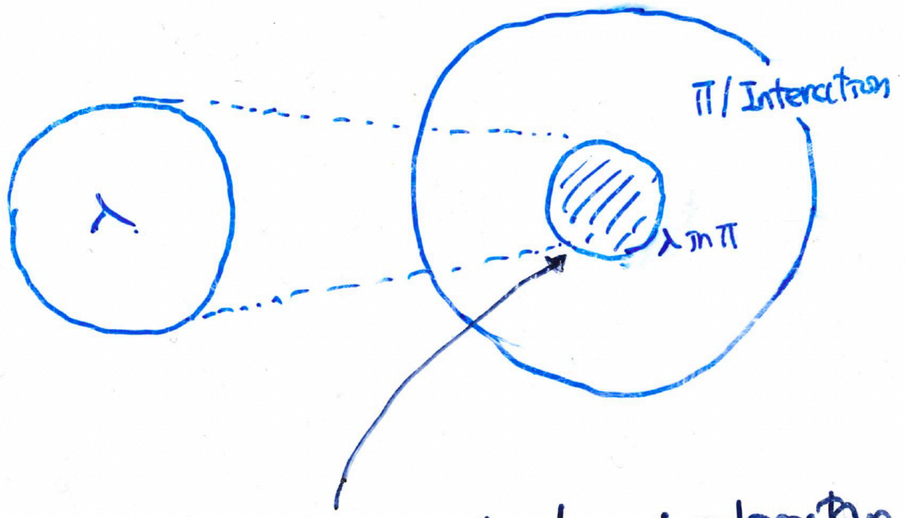
with $[\![x=N]\!] \stackrel{\text{def}}{=} !x(u').[\![N]\!]_{u'}.$

Prologue (1)

- "Functions as Processes" : [Milner 90]
  cf. [Girard 87].

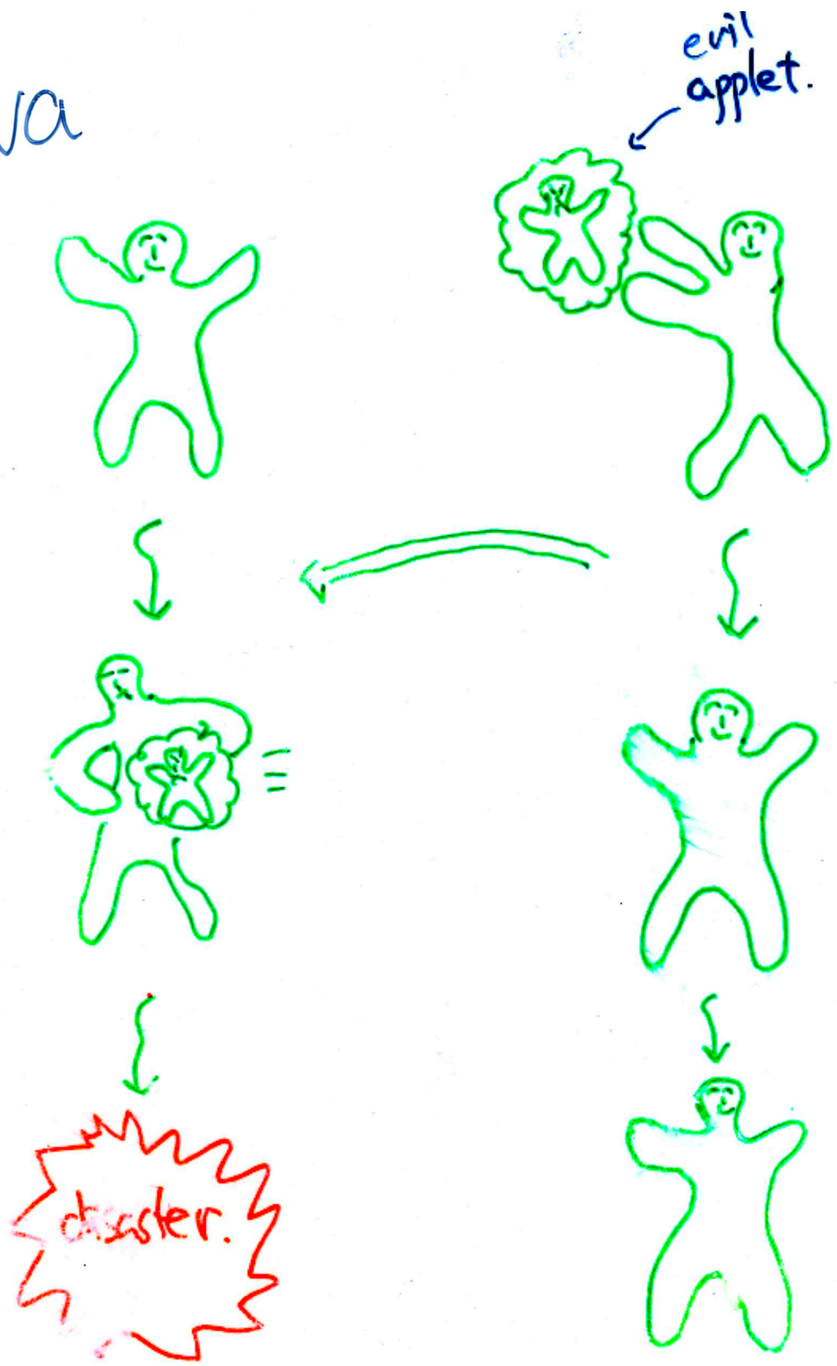
Functions

Interactions

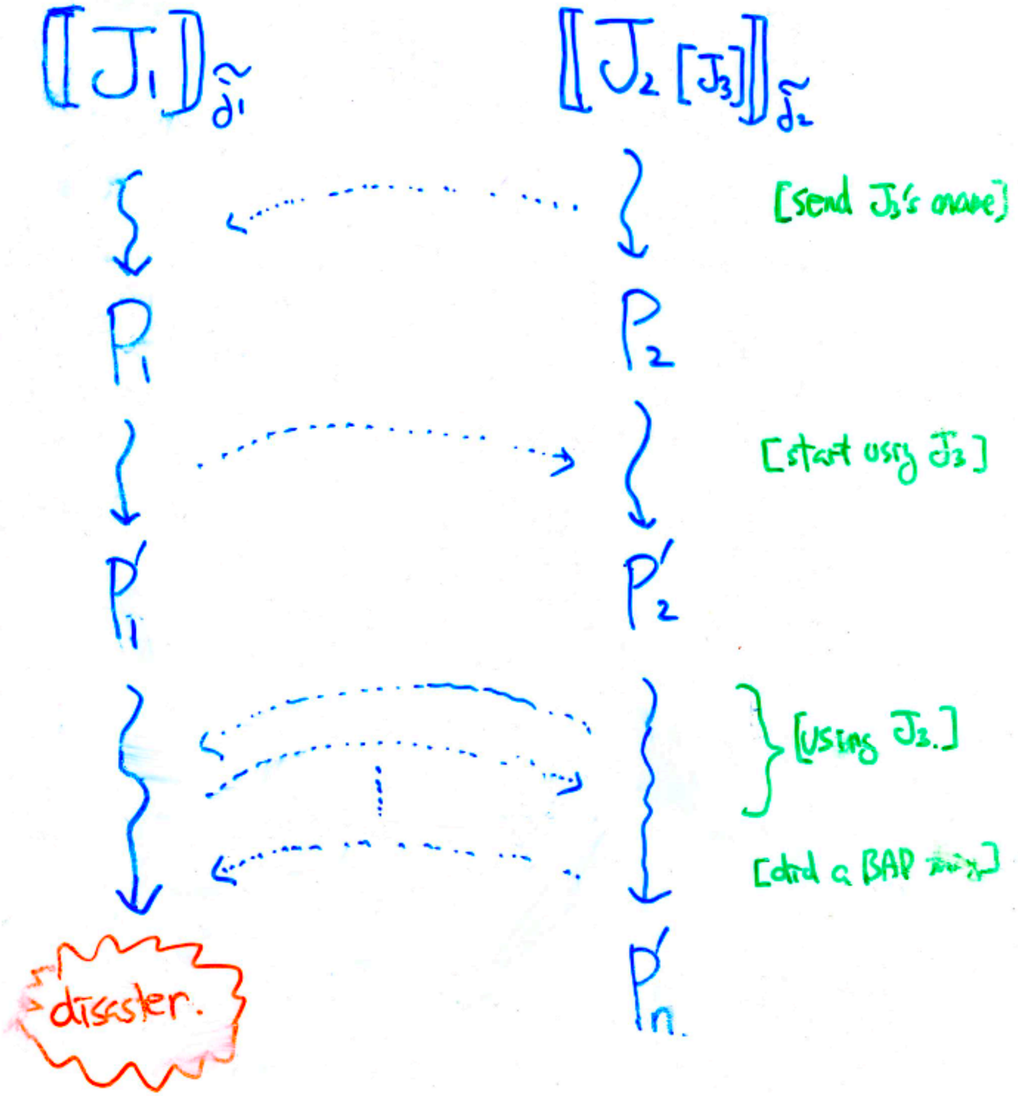


☆ What are the class of interactive behaviours, or Rules of Interaction, of λ-agents ?

Java

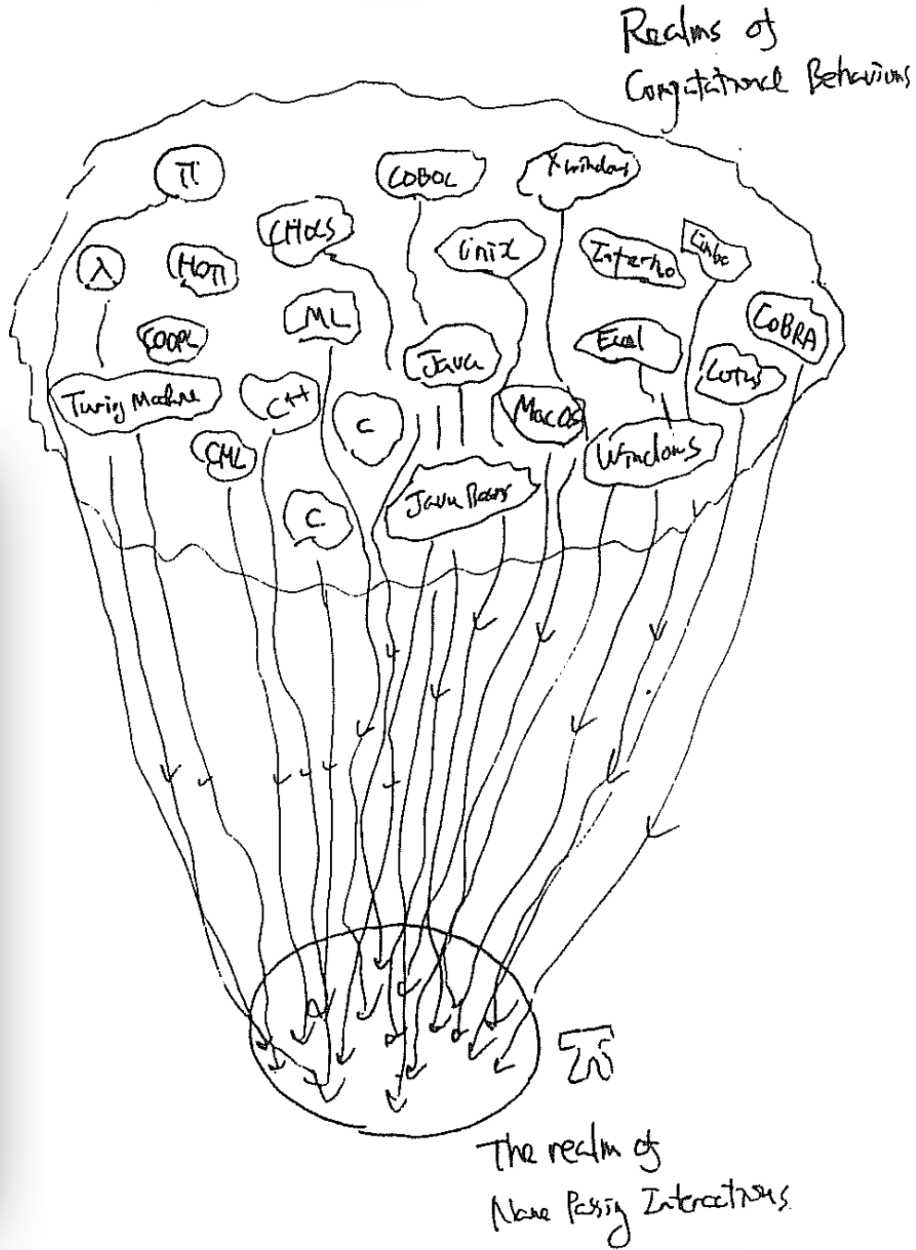


Java in π



* Examples of Representable Computation.

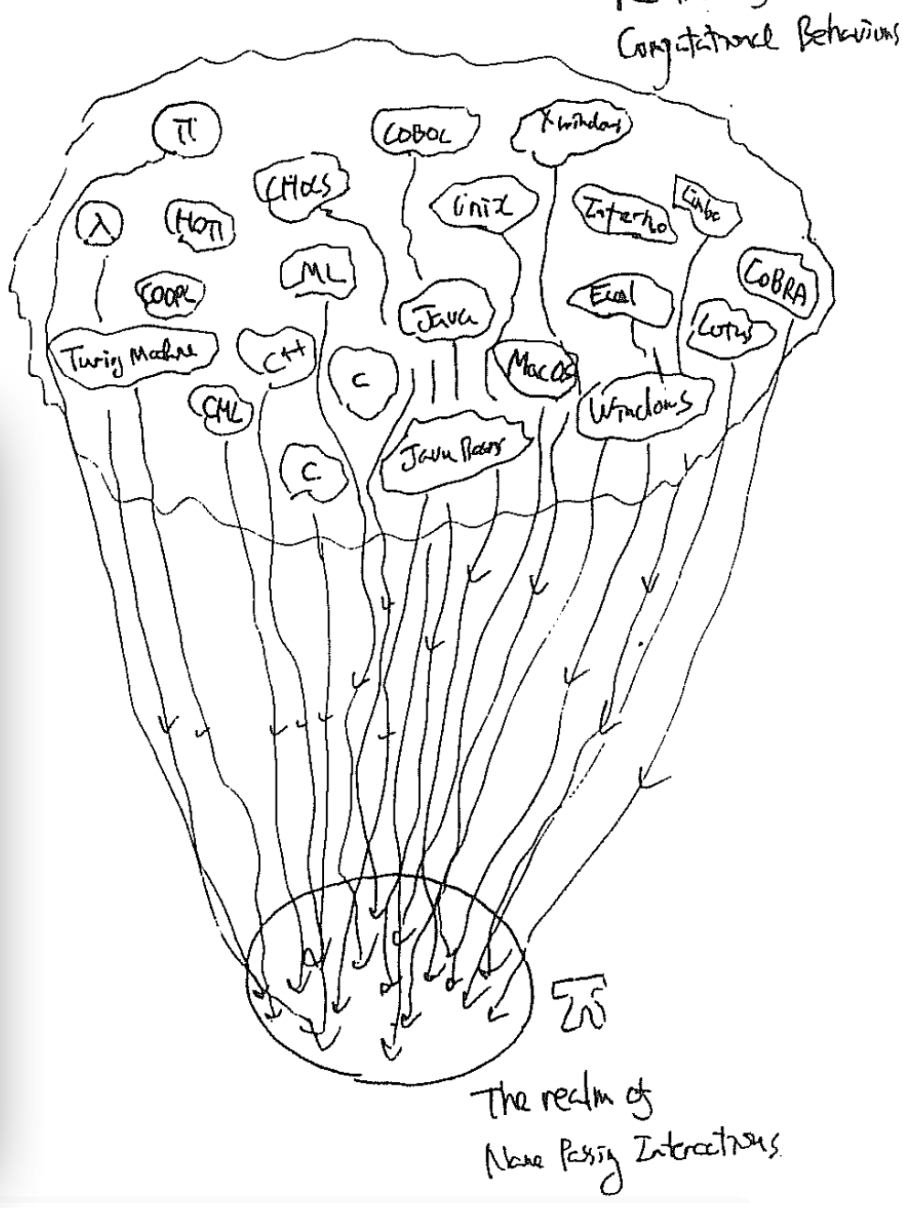
- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner92, ...]
- Proof Nets [Bellin and Scott93]
- Arbitrary 'constant' interaction [HYS4]
- strategies on Games [HO95]

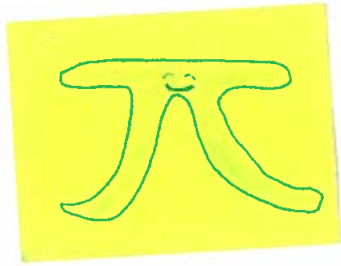


* Examples of Representable Computation.

Operationally Sound but NOT Fully Abstract

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner92, ...]
- Proof Nets [Bellin and Scott93]
- Arbitrary 'constant' interaction [HYS4]
- Strategies on Games [HO95]
- ⋮



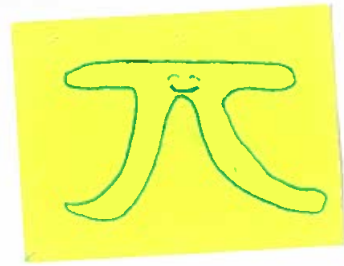


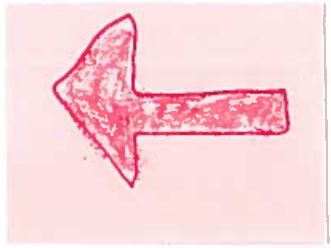
$M \simeq N$

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

Conextual Congruence

Contextual Congruence



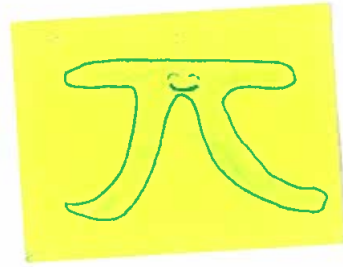
$M \simeq N$  $[\![M]\!] \simeq [\![N]\!]$

Conextual Congruence

Contextual Congruence

$C[P]\Downarrow_a$ iff $C[Q]\Downarrow_a$

$C[[\![M]\!]]\Downarrow_a$ iff $C[[\![N]\!]]\Downarrow_a$

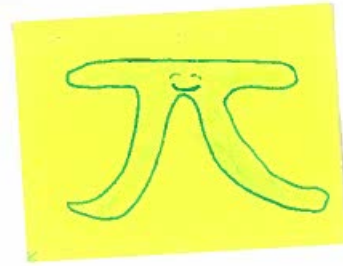


$M \simeq N$ 

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

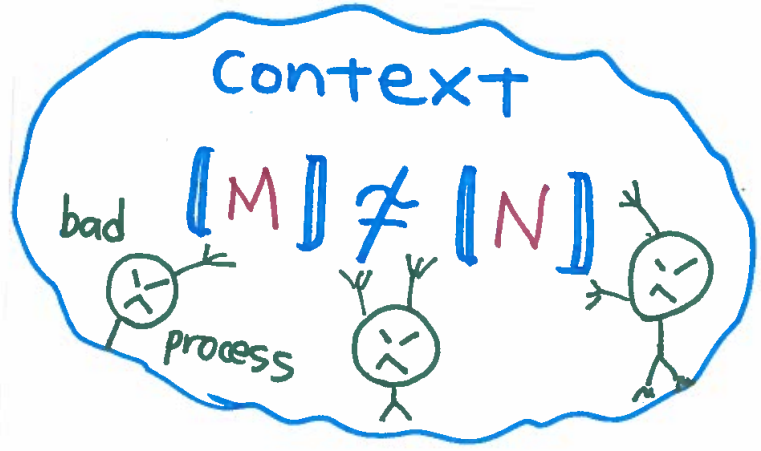
$C[P]\Downarrow_a$ iff $C[Q]\Downarrow_a$

$C[\![M]\!]\Downarrow_a$ iff $C[\![N]\!]\Downarrow_a$



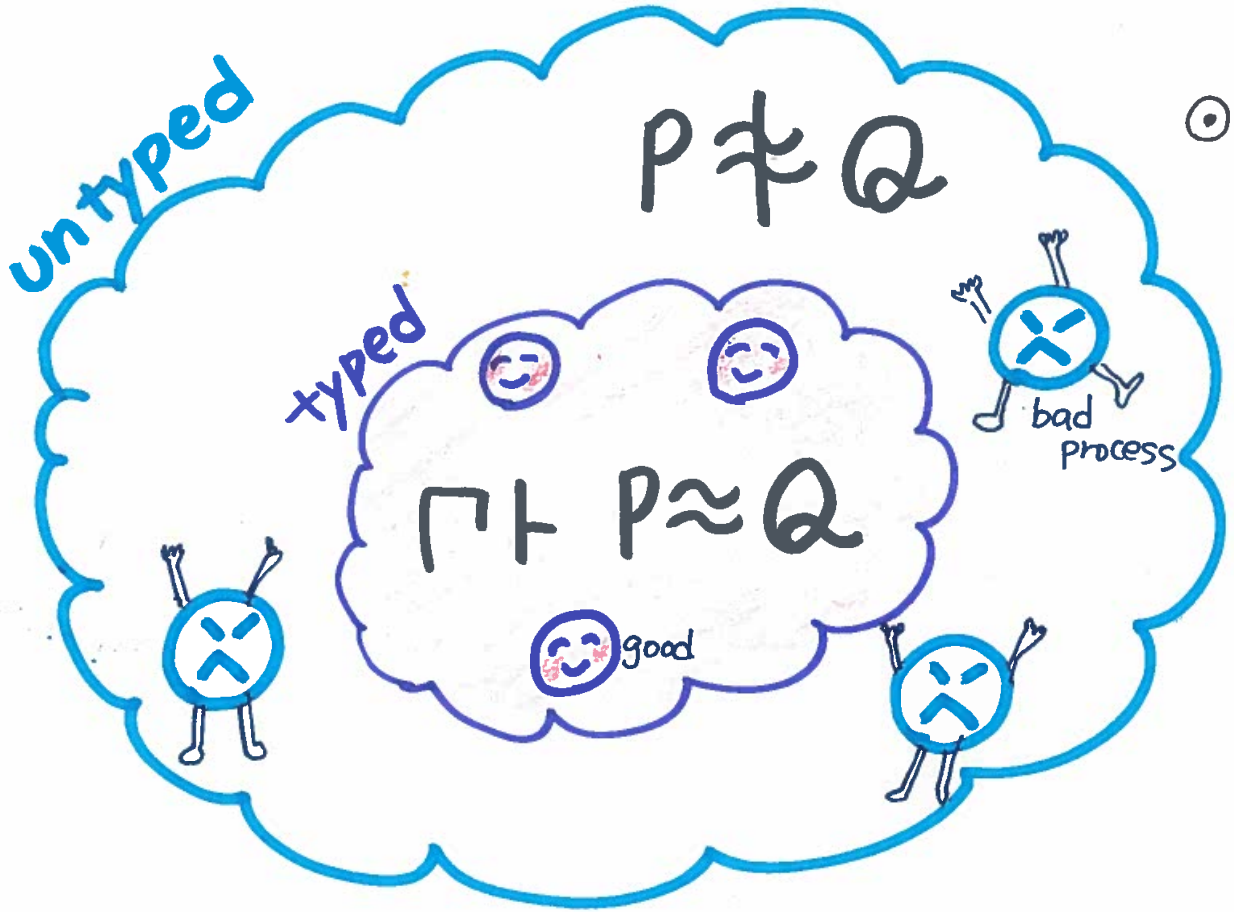
$M \simeq N$ 

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



Typed Semantics in π 1991→

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



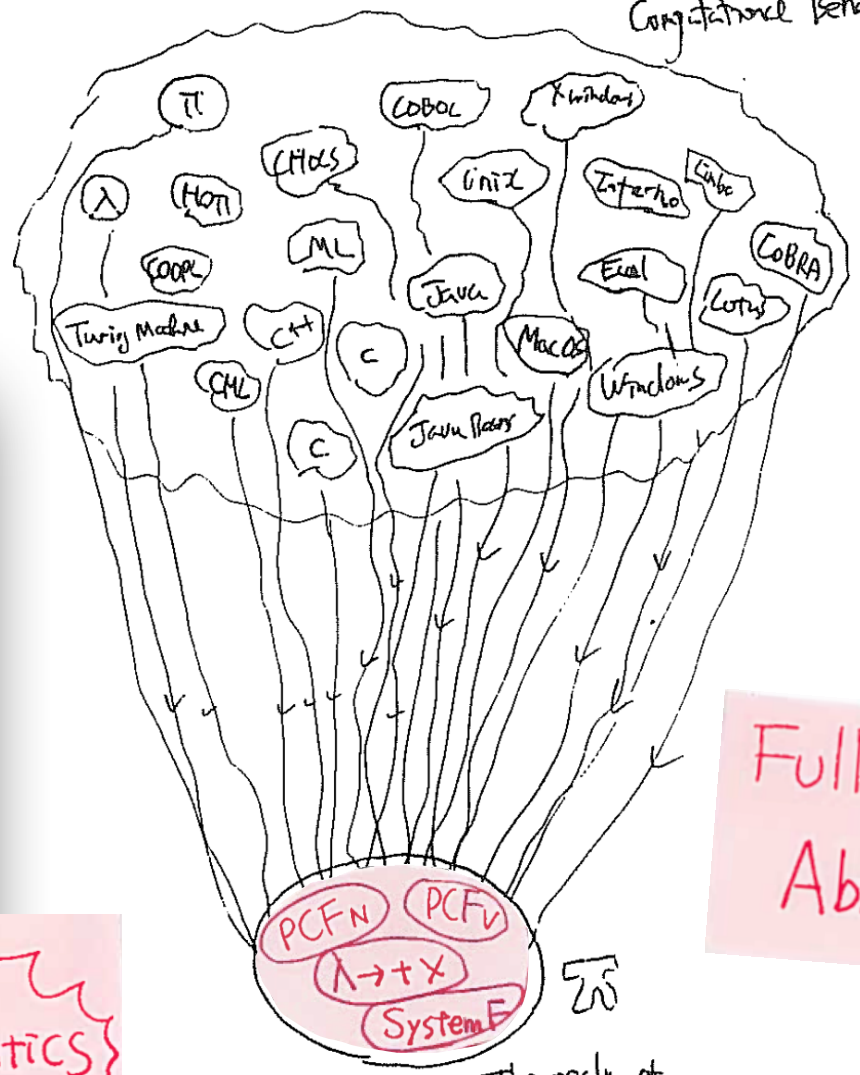
- ⊙ Correctness of Encoding
- ⊙ Limit environment Γ
  ⇒ Equate more processes
- ⊙ Compositional

* Examples of Representable Computation.

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [HYS4]
- strategies on Games [HO95]
- ⋮

Game Semantics

Realms of Computational Behaviors



The realm of Name Passing Interactions

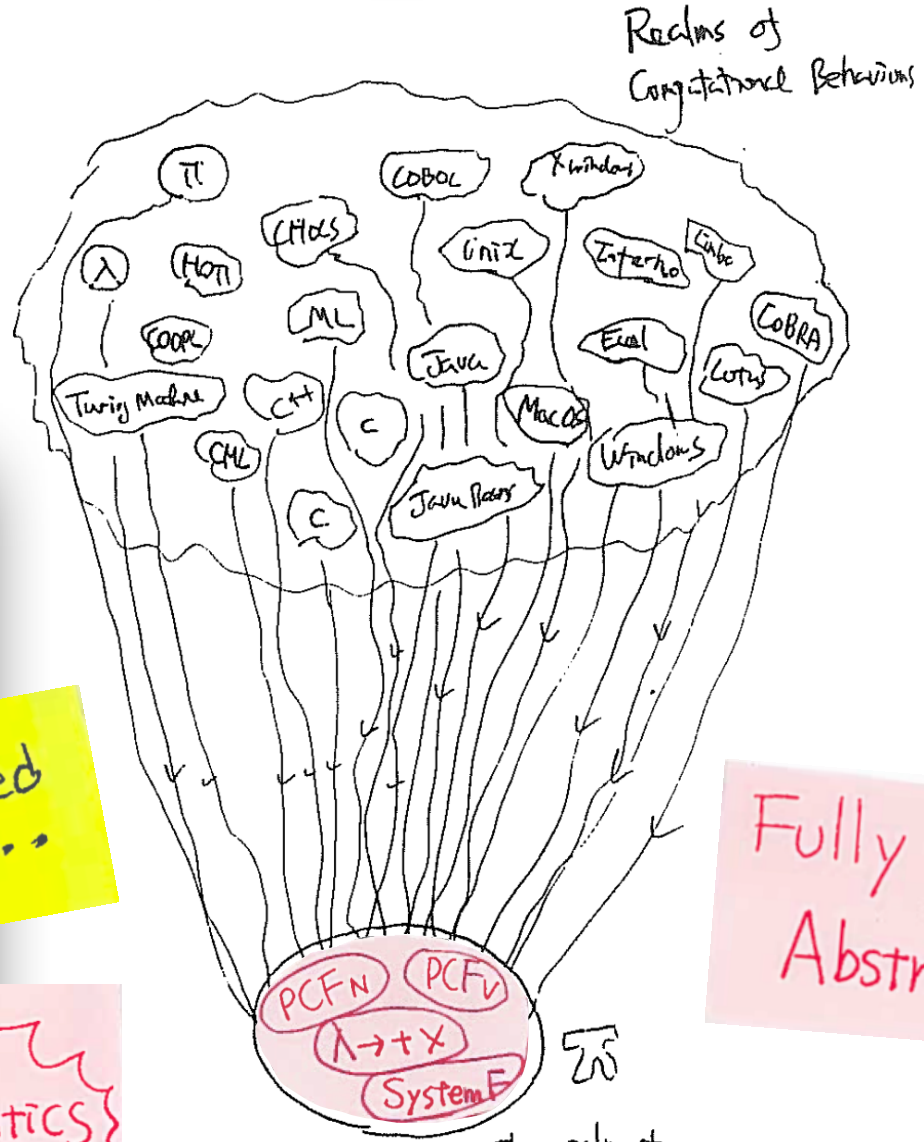
Fully Abstract

* Examples of Representable Computation.

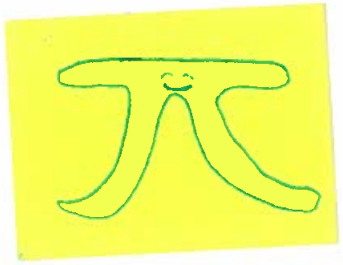
- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [
- Strategies on Games [HO95]

Complicated ...

Game Semantics

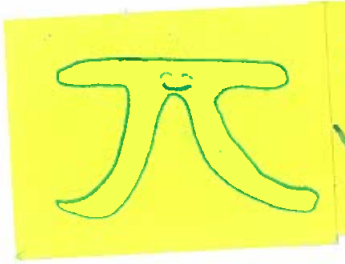


Fully Abstract



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

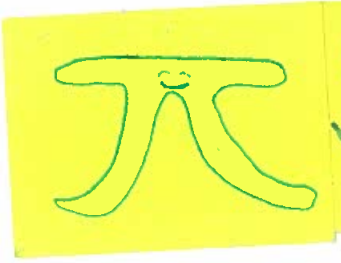
System F



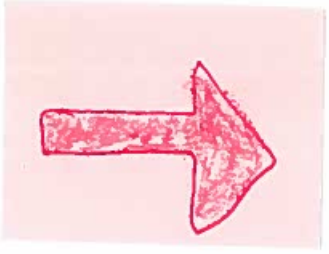
Session $\Pi\Pi$

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

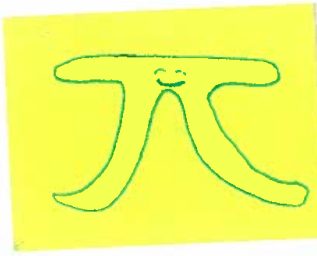
System F



Session

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

System F



Session Π

$M \simeq N$

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

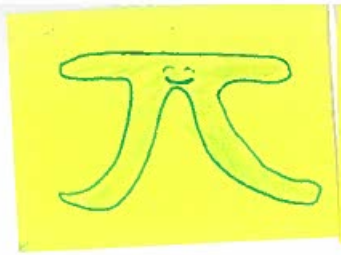
$[\![P]\!] \simeq [\![Q]\!]$

$P \simeq Q$

Reverse

New

System F



Session

$M \simeq N$

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

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

$P \simeq Q$

Reverse

New

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

$X_1, .., X_k \; ; \; a_1 : A_1, \ldots, a_n : A_n \vdash P :: b : A$

Process

Poly Vars

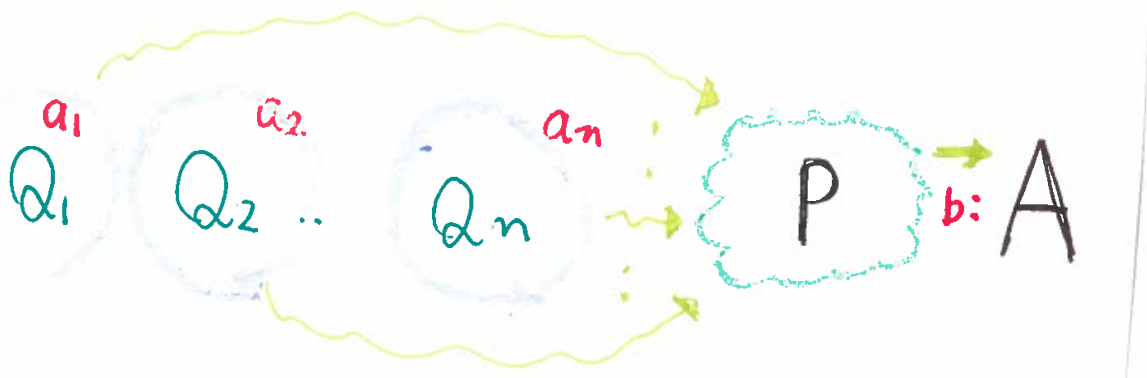
name

Type

name

type

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



Judgement

$X_1, .., X_k \; ; \; a_1 : A_1, \ldots, a_n : A_n \vdash P :: b : A$

Process

Poly Vars

name

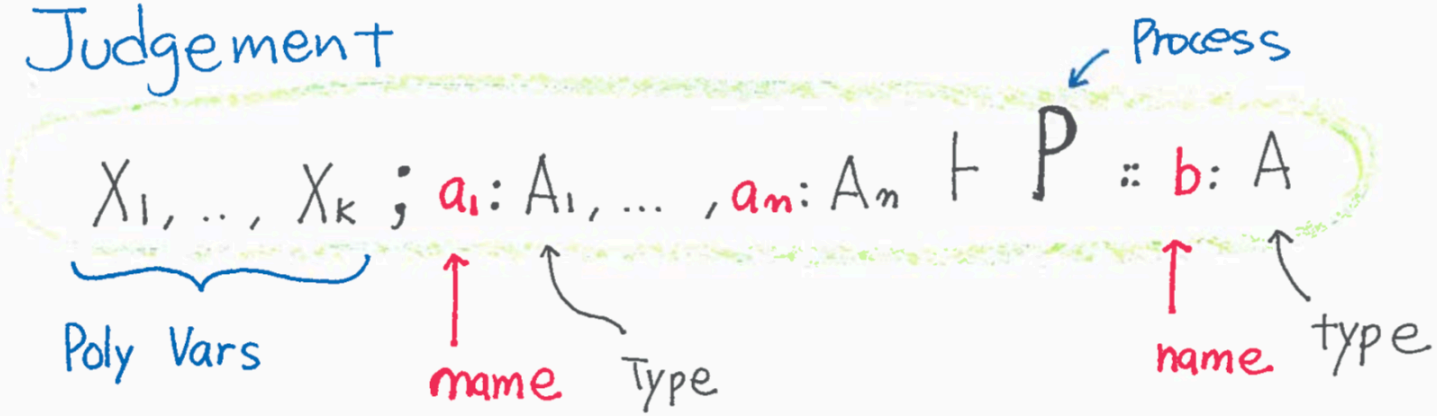
Type

name

type

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

Judgement



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

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

Input

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

output

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

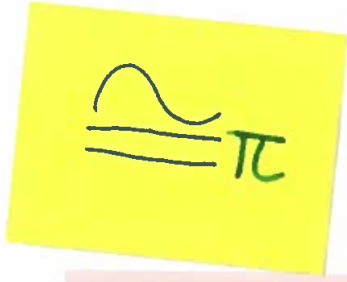
Polymorphic Session

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



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

output

Barbed Congruence

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

Linear F zhao, Zhang, Zdancewic 2010

Types

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

Terms

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

Encoding: from  to  Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

Intro ⇒ Right / Elim ⇒ Left + Cut + Identity

Encoding: from λ to π

Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

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

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

a ↑ name

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

$$[\![\langle\rangle]\!]_a = 0 \qquad [\![\lambda x.M]\!]_a = a(x).\, [\![M]\!]_a$$

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

The π-calculus as a Descriptive Tool

λ in π

Milner's Encoding 1991

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

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

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

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

Encoding: from λ to π

Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

Intro ⇒ Right / Elim ⇒ Left + Cut + Identity

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

↑ name

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

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

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

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

λ in π

$[\![x]\!]_u \stackrel{def}{=} \overline{x}(u).$

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

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

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

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

Type

$$[\![x\langle B\rangle.\, P]\!] = [\![P]\!]\{x[B]/x\}$$

Application of B to 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

Definability

Inverse

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

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

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

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

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

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

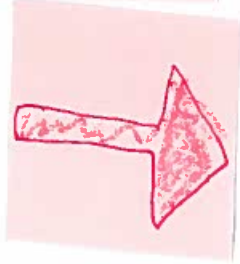
$$[\![(\!|\, P \,|\!)]\!]_z \cong P$$

Full Abstraction

$$[\![M]\!]_z \cong [\![N]\!]_z$$

$$M \cong N$$

$$(\!|\, P \,|\!) \cong (\!|\, Q \,|\!)$$

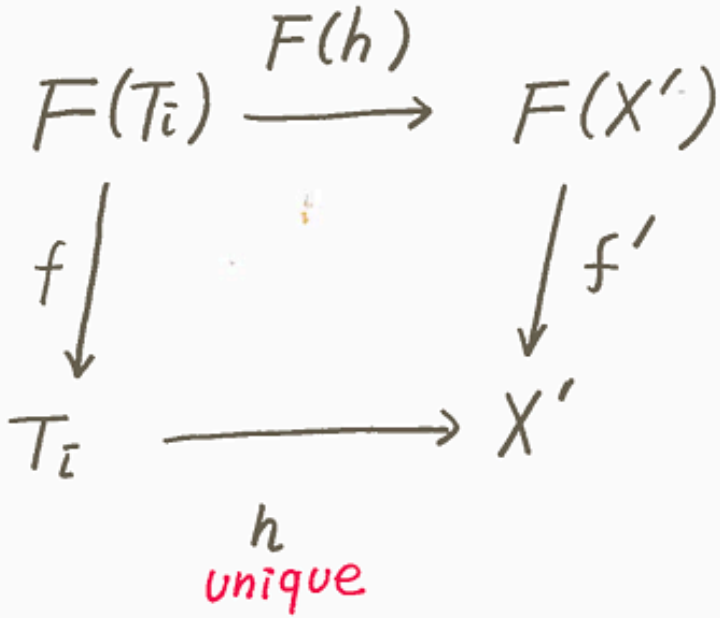


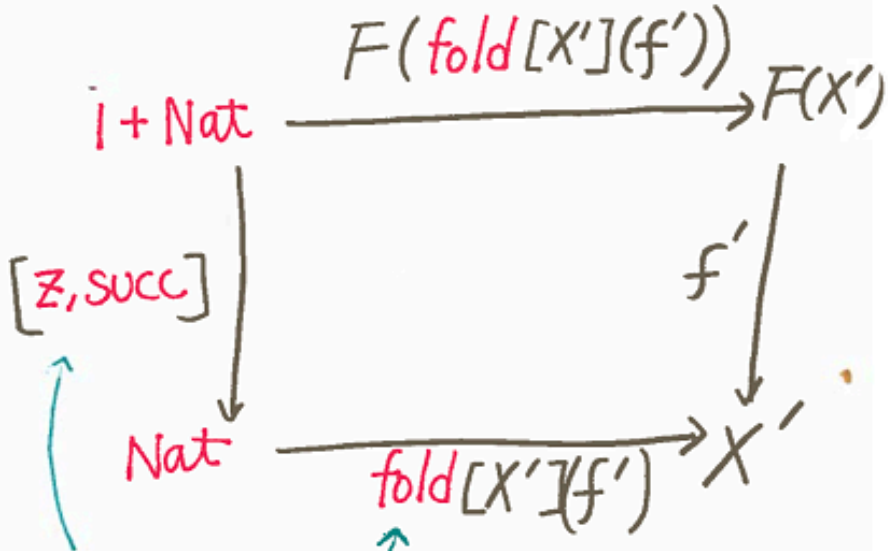
$$P \cong Q$$

Application 1 Inductive and Coinductive Types

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

as Initial / Final (Co)Algebra

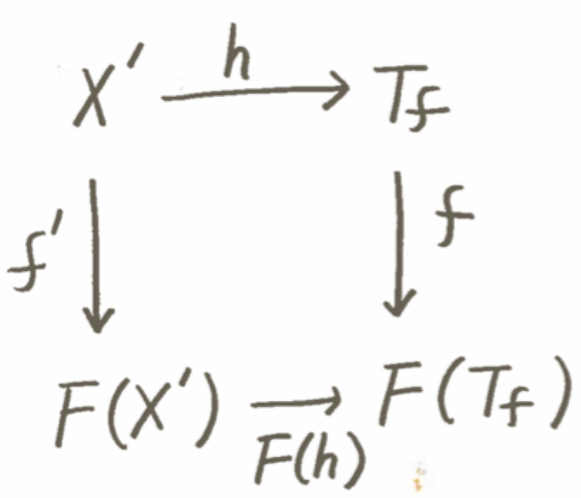


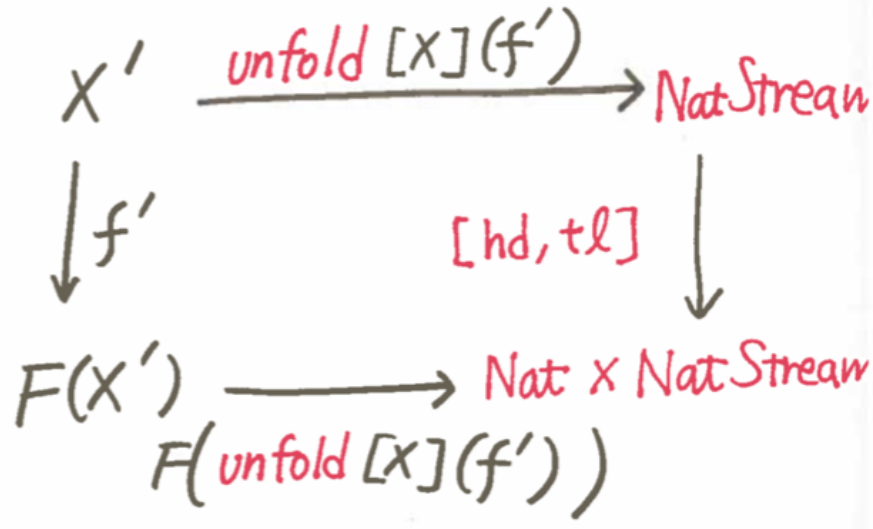


can be defined in System F and fold is unique for all inductive types

Inductive

Coinductive Types





Question Sess Poly π is Expressive Enough?

Theorems

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

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

Application (2) HOπ + PRocess Monad

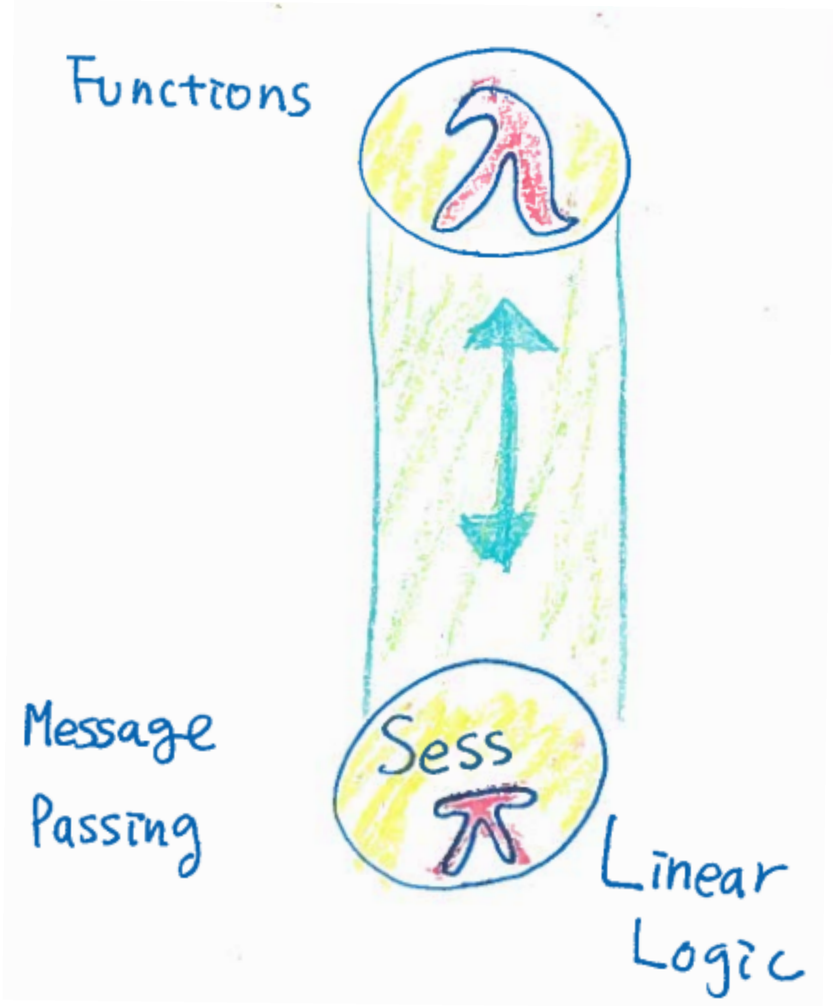
- Full Abstraction / Inverse $x\langle M\rangle.P$
- Strong Normalisation via Strong Normalisation

HOπ : λ

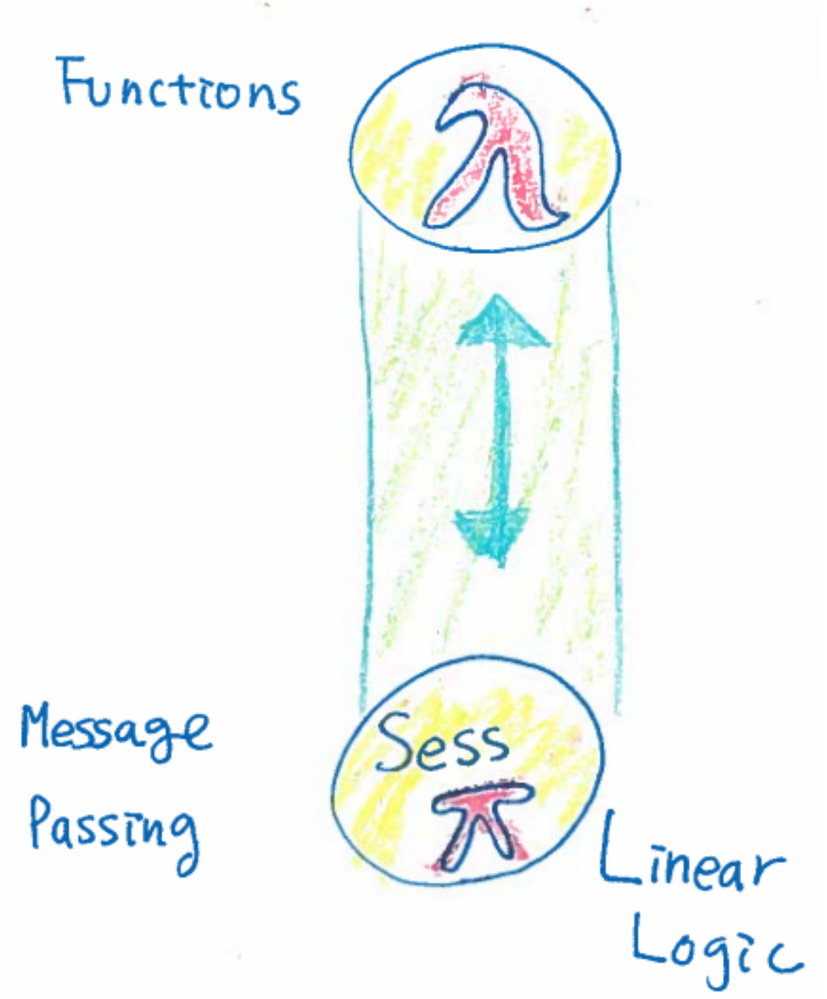
$$P \to Q \quad \Rightarrow \quad (\!|P|\!) \to^{+} (\!|Q|\!)$$

cf. [CPPT'13] logical relation

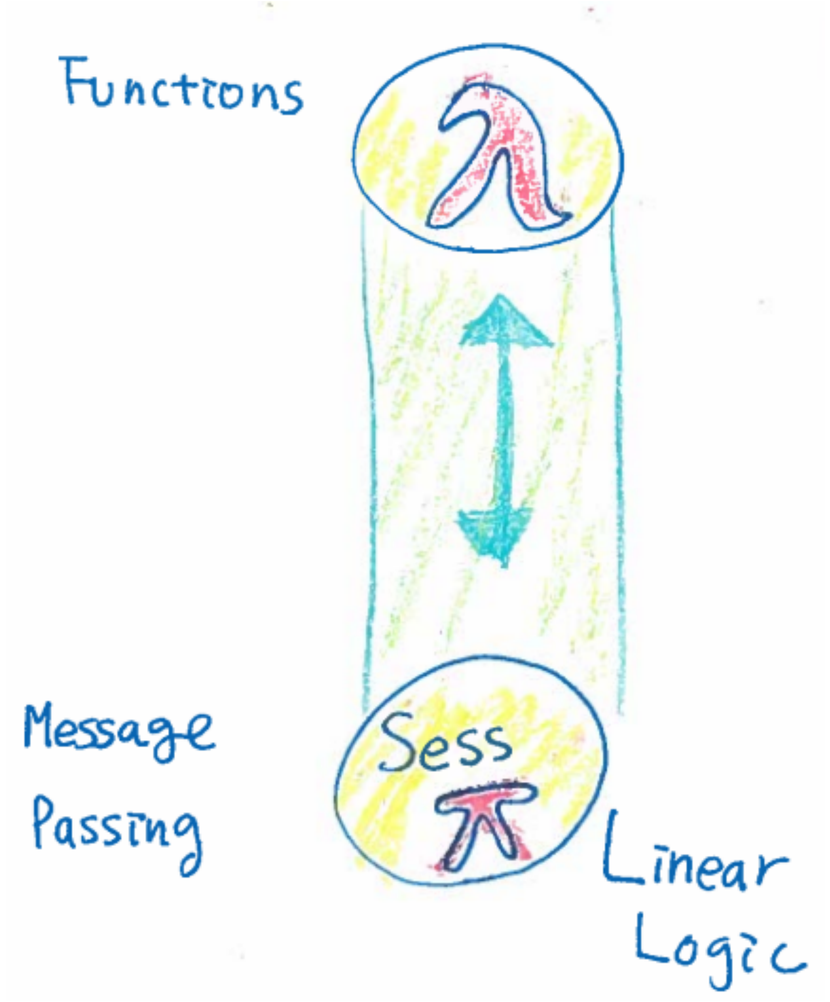
Summary



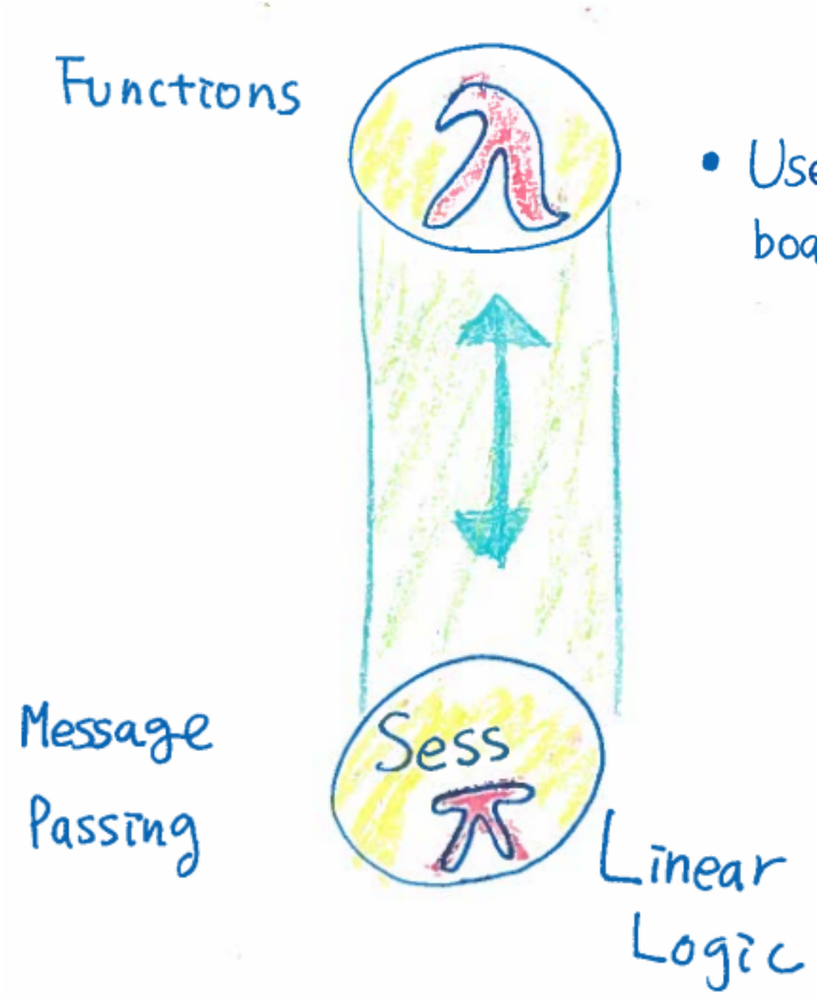
Summary



Summary

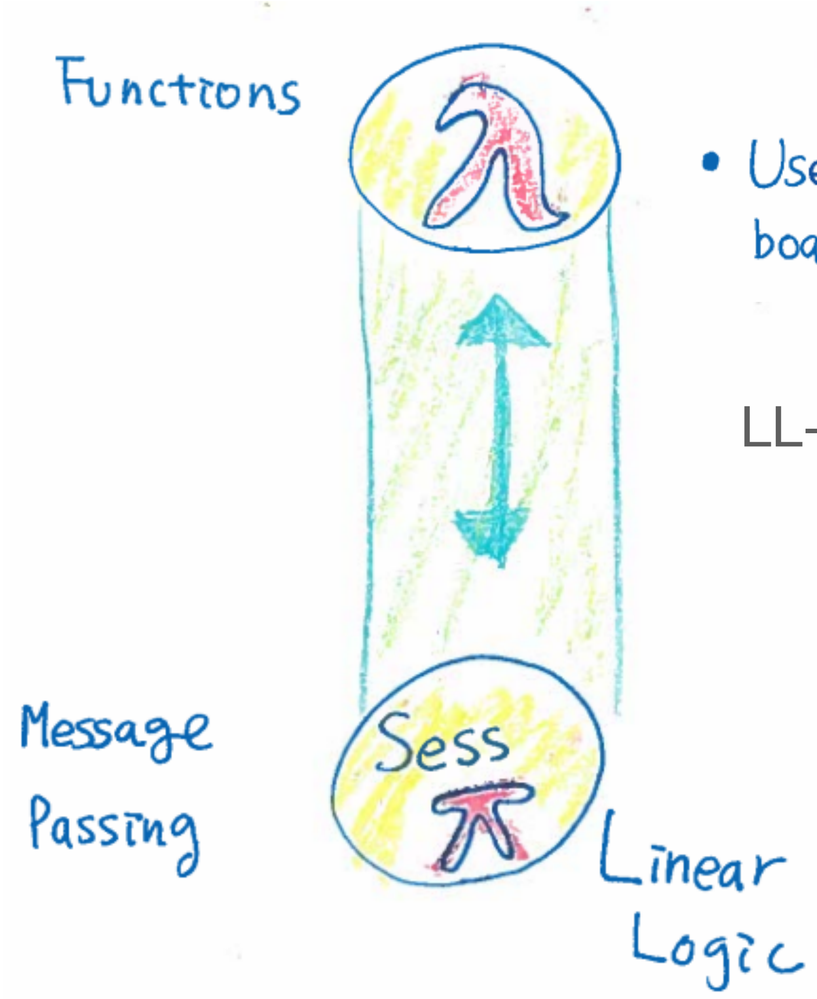


Summary



- LL-based Use Sess π to articulate boarder computations

Summary



LL-based

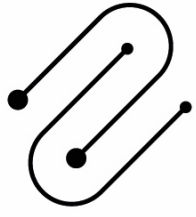
- Use Sess π to articulate boarder computations

LL-Based Session Types (Too Many to List)

- Balzer and Pfenning, Manifest sharing with session types, ICFP'17
- Rocha and Caires, Propositions-as-types and shared state, ICFP'21
- Zien et al, Client-server sessions in linear logic, ICFP'21
- Frumin et al, A propositions-as-sessions interpretation of bunched implications in channel-based concurrency, OOPSLA'22
- Jacobs et al., Multiparty GV: Functional Multiparty Session Types With Certified Deadlock Freedom

Termination in Concurrency, Revisited

Work by:
Joseph Paulus (University of Groningen, NL)
Daniele Nantes-Sobrinho (Imperial College, UK)
Jorge A. Pérez (Univ. of Groningen)



UNIFYING
C•RRECTNESS FOR
C•MMUNICATING
S•FTWARE

Principles and Practice of Declarative Programming (PPDP23)

Termination

- A term is terminating if all its reduction sequences are finite.
- A central property in sequential programming models.

Termination

- A term is terminating if all its reduction sequences are finite.
- A central property in sequential programming models.
- Also important in concurrency, and for **message-passing programs**.
- We want: processes with **infinite** sequences of **observable** actions, intertwined with **finite** sequences of **internal** actions (reductions).

Termination

- A term is terminating if all its reduction sequences are finite.
- A central property in sequential programming models.
- Also important in concurrency, and for **message-passing programs**.
- We want: processes with **infinite** sequences of **observable** actions, intertwined with **finite** sequences of **internal** actions (reductions).

Our work

- Termination of π-calculus processes.
  Infinite behaviors arise from server processes (replicated inputs).

$$!x(y).x\langle y\rangle.0 \mid x\langle w\rangle.0 \longrightarrow !x(y).x\langle y\rangle.0 \mid x\langle w\rangle.0 \mid 0 \longrightarrow \cdots$$

Termination

- A term is terminating if all its reduction sequences are finite.
- A central property in sequential programming models.
- Also important in concurrency, and for **message-passing programs**.
- We want: processes with **infinite** sequences of **observable** actions, intertwined with **finite** sequences of **internal** actions (reductions).

Our work

- Termination of π-calculus processes.
  Infinite behaviors arise from server processes (replicated inputs).

$$!x(y).x\langle y\rangle.0 \mid x\langle w\rangle.0 \longrightarrow !x(y).x\langle y\rangle.0 \mid x\langle w\rangle.0 \mid 0 \longrightarrow \cdots$$

- Several different **type systems**: how do they relate to each other?
- We **formally compare** two representative type systems.
  Approach: Contrast the classes of (session) processes they can type.

Approach and Results

The Class $\mathcal{S}$

- Session typed π-calculus by Vasconcelos
- Very flexible and expressive
- Good baseline for comparisons

Approach and Results

The Class $\mathcal{S}$

- Session typed π-calculus by Vasconcelos
- Very flexible and expressive
- Good baseline for comparisons

The Class $\mathcal{L}$

- Follows 'propositions-as-sessions' by Caires & Pfenning
- Termination via Curry-Howard

Approach and Results

The Class $\mathcal{S}$

- Session typed π-calculus by Vasconcelos
- Very flexible and expressive
- Good baseline for comparisons

The Class $\mathcal{L}$

- Follows 'propositions-as-sessions' by Caires & Pfenning
- Termination via Curry-Howard

The Class $\mathcal{W}$

- Follows weight-based type systems by Deng & Sangiorgi
- Proof by well-founded measures

Approach and Results

The Class $\mathcal{S}$

- Session typed π-calculus by Vasconcelos
- Very flexible and expressive
- Good baseline for comparisons

The Class $\mathcal{L}$

- Follows 'propositions-as-sessions' by Caires & Pfenning
- Termination via Curry-Howard

The Class $\mathcal{W}$

- Follows weight-based type systems by Deng & Sangiorgi
- Proof by well-founded measures

- Three process models (π_{S}, π_{W}, π_{DILL}) with different syntax/types.
- Classes $\mathcal{W}$ and $\mathcal{L}$ defined using **typed encodings**. We have $\mathcal{W} \subset \mathcal{S}$ and $\mathcal{L} \subset \mathcal{S}$.
- Main results: $\mathcal{L} \subset \mathcal{W}$ and $\mathcal{W} \not\subset \mathcal{L}$.

Main Results

$\mathcal{L} \subset \mathcal{W}$

- Every (terminating) process in the logic-based approach is typable using the weight-based approach.
- Key idea: For every $P \in \mathcal{L}$, extract a **weight function** to enable typability for P in the weight-based system (up to a typed encoding).

$\mathcal{W} \not\subset \mathcal{L}$

- By counter-example: there are terminating processes typable using weights, but not typable in the Curry-Howard correspondence.
- The weight-based approach captures **more server behaviors** than those induced by linear logic (copying semantics for $!A$).

Approach and Results

The Class $\mathcal{S}$

- Session typed π-calculus by Vasconcelos
- Very flexible and expressive
- Good baseline for comparisons

The Class $\mathcal{L}$

- Follows 'propositions-as-sessions' by Caires & Pfenning
- Termination via Curry-Howard

The Class $\mathcal{W}$

- Follows weight-based type systems by Deng & Sangiorgi
- Proof by well-founded measures

Approach and Results

The Class $\mathcal{S}$

- Session typed π-calculus by Vasconcelos
- Very flexible and expressive
- Good baseline for comparisons

The Class $\mathcal{L}$

- Follows 'propositions-as-sessions' by Caires & Pfenning
- Termination via Curry-Howard

The Class $\mathcal{W}$

- Follows weight-based type systems by Deng & Sangiorgi
- Proof by well-founded measures

- Three process models (π_{S}, π_{W}, π_{DILL}) with different syntax/types.
- Classes $\mathcal{W}$ and $\mathcal{L}$ defined using **typed encodings**. We have $\mathcal{W} \subset \mathcal{S}$ and $\mathcal{L} \subset \mathcal{S}$.
- Main results: $\mathcal{L} \subset \mathcal{W}$ and $\mathcal{W} \not\subset \mathcal{L}$.