Session TYPES

AS A

Descriptive π Tool

for Distributed PROTOCOL

Nobuko Yoshida & Raymond Hu

25th Oct 2017

Awards

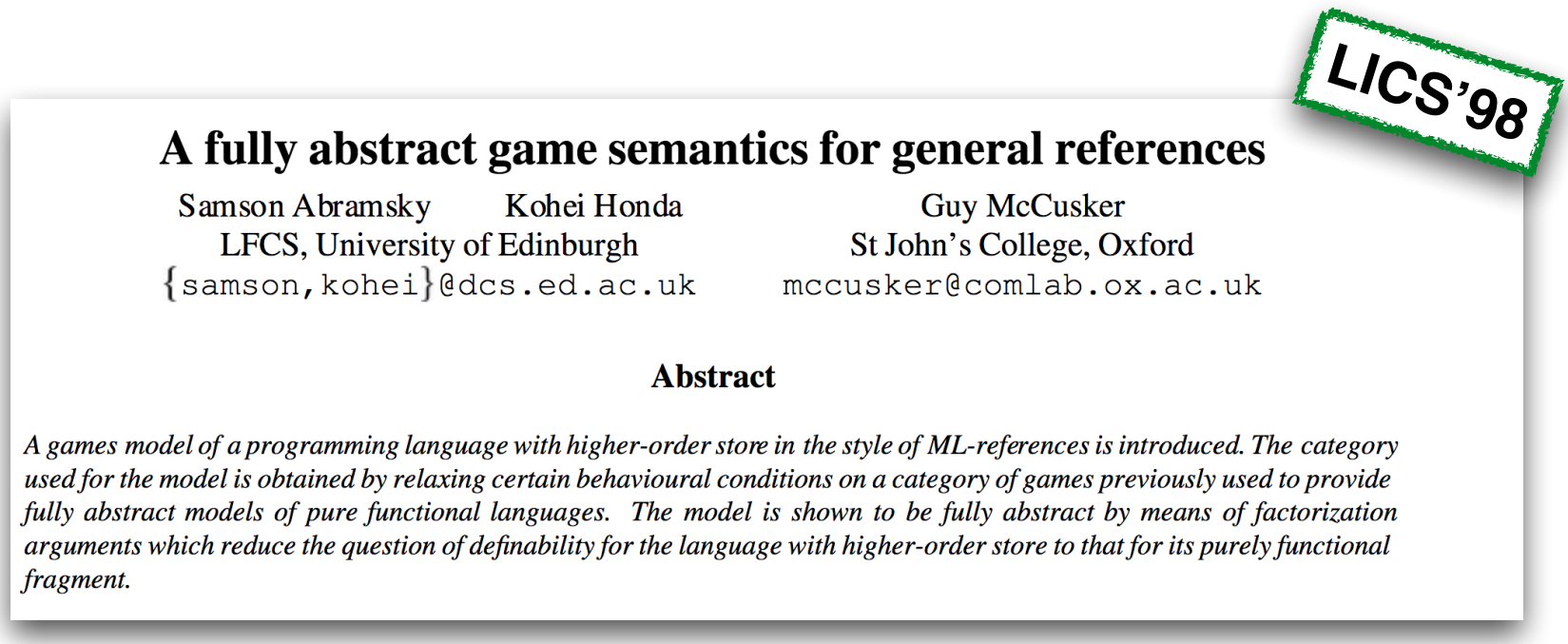

ETAPS 2019
TEST-OF-TIME
AWARD
Language primitives and type discipline for structured communication-based programming' by Kohei Honda, Vasco T. Vasconcelos and Makoto Kubo

Awards

POPL 2008 MOST INFLUENTIAL PAPER AWARD

Awards


LICS 2018 TEST OF TIME AWARD

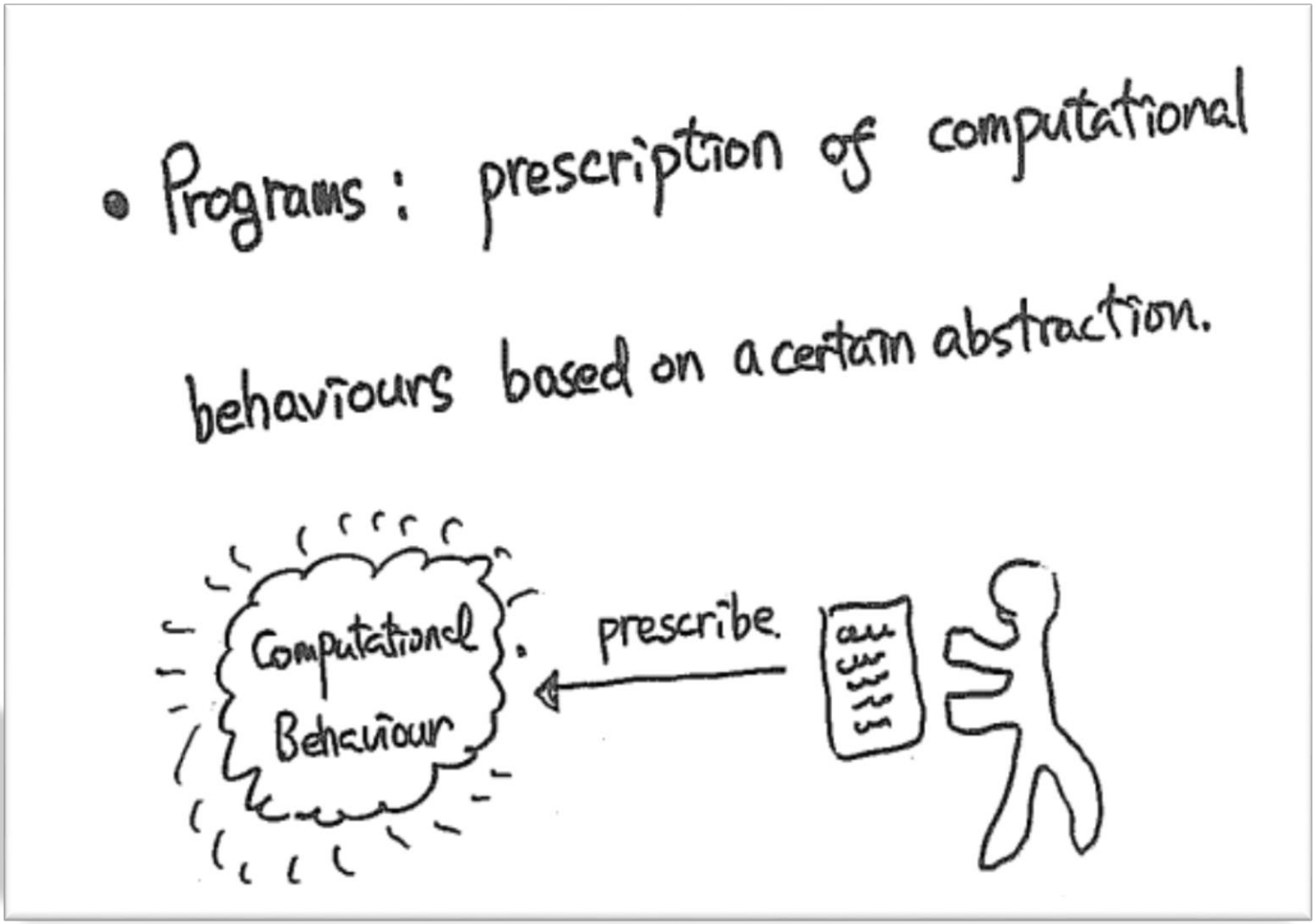

LICS'98
A fully abstract game semantics for general references
Samson Abramsky
Kohei Honda
LFCS, University of Edinburgh
{samson,kohei}@dcs.ed.ac.uk
Guy McCusker
St John's College, Oxford
mccusker@comlab.ox.ac.uk
Abstract
A games model of a programming language with higher-order store in the style of 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:

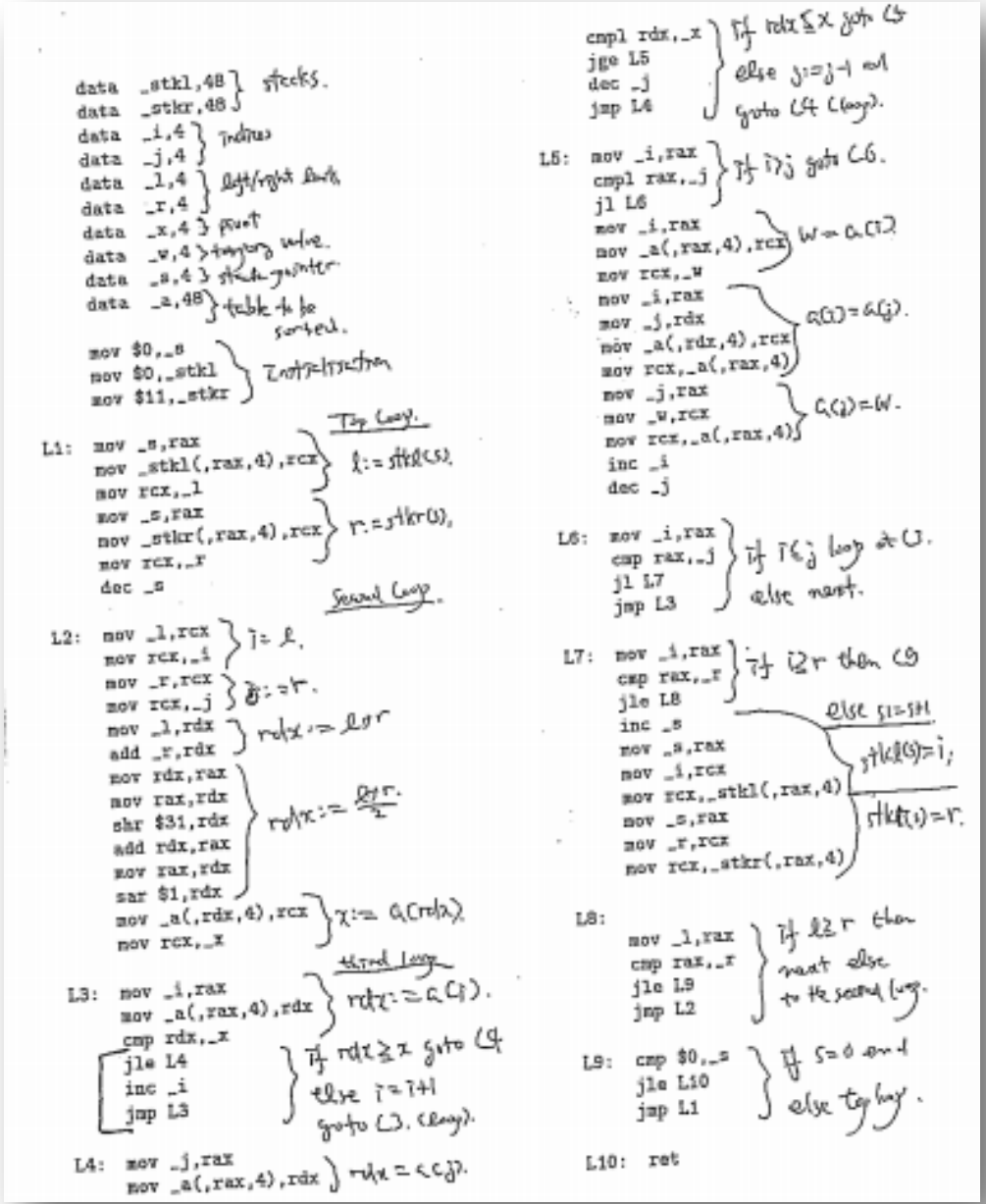
  Imperative: sequential composition, if-then-else,

  while, procedures, modules, ...

  Functional: application, product, union,

  recursion, modules, ...

UNSTRUCTURED:



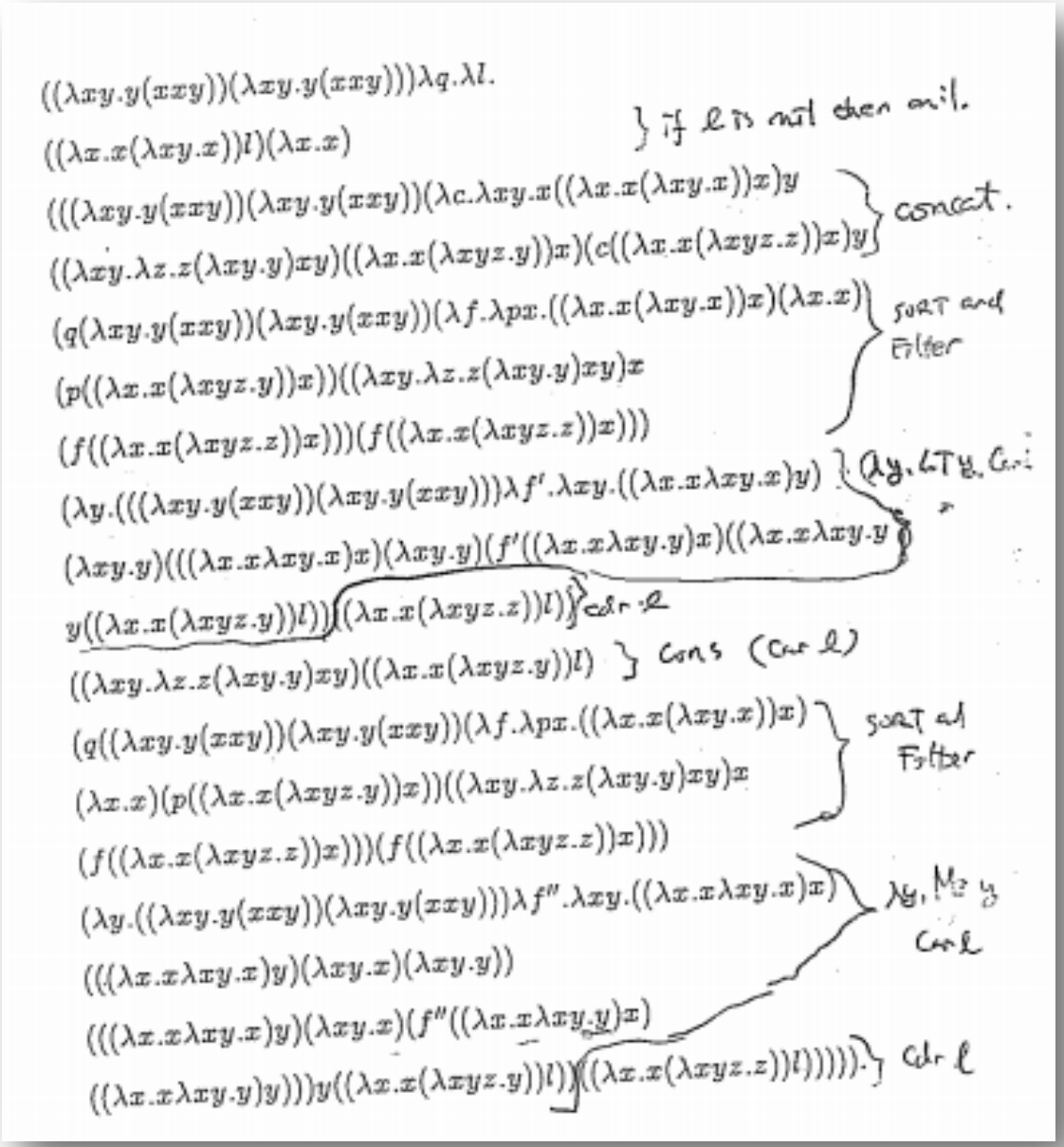
STRUCTURED:

```
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 π-calculus as a Descriptive Tool

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

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

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

λ in π

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

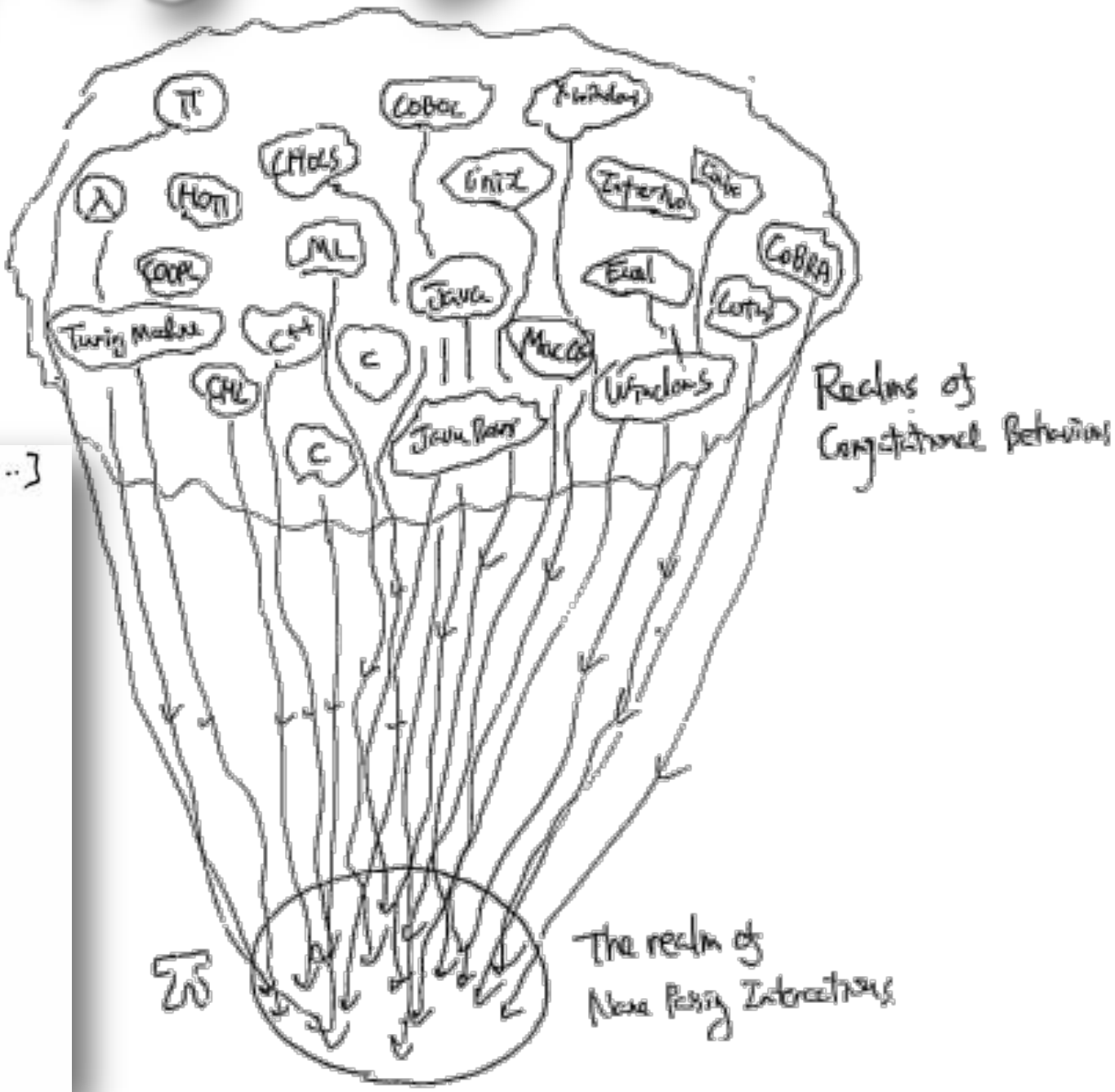
$[\![\lambda x.M]\!]_u \stackrel{\text{def}}{=} u(x u').[\![M]\!]_{u'}.$

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

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 [HYS4]
- strategies on Games [HO95]

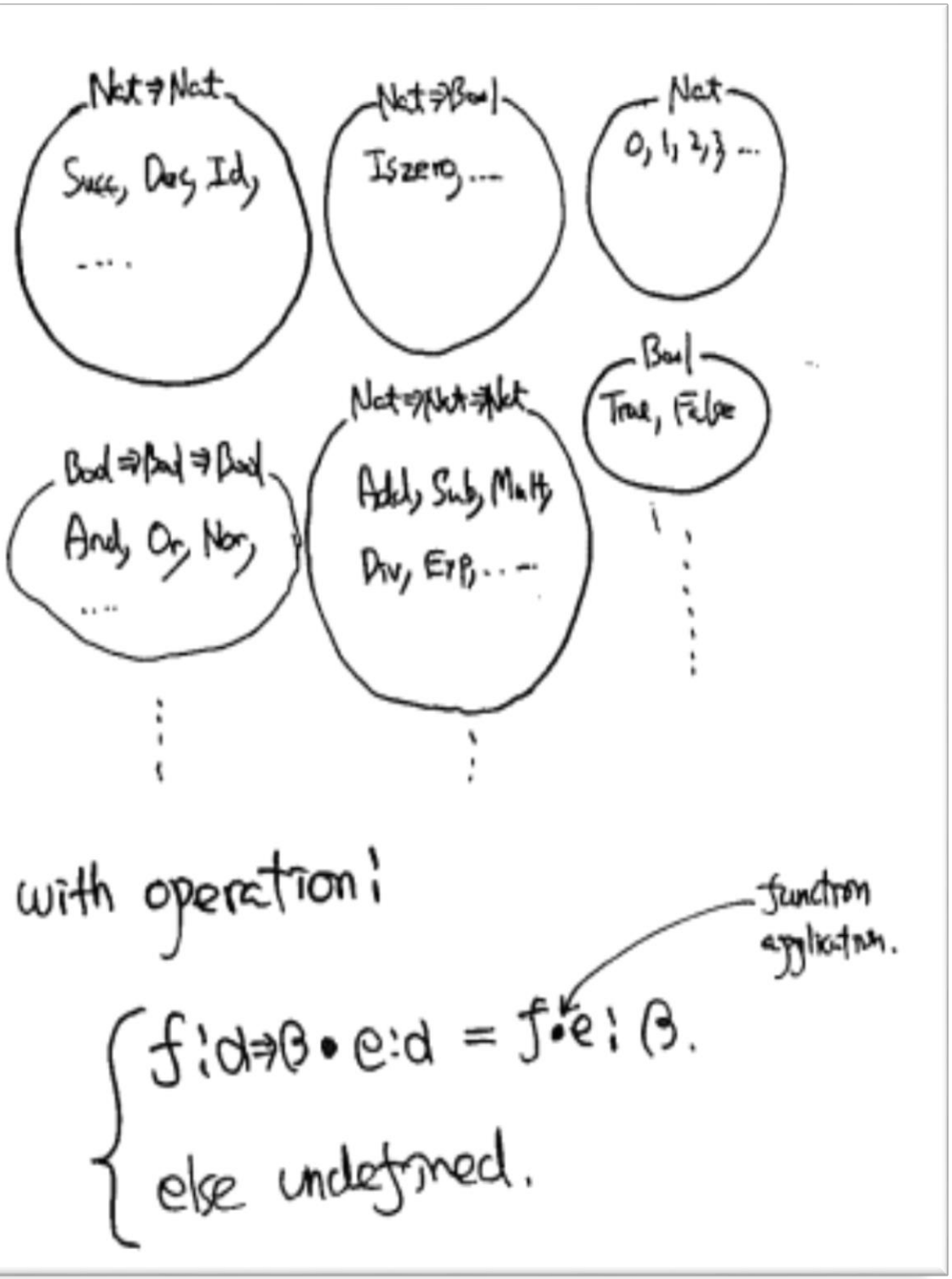


The Role of Types in π-calculus

- (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?

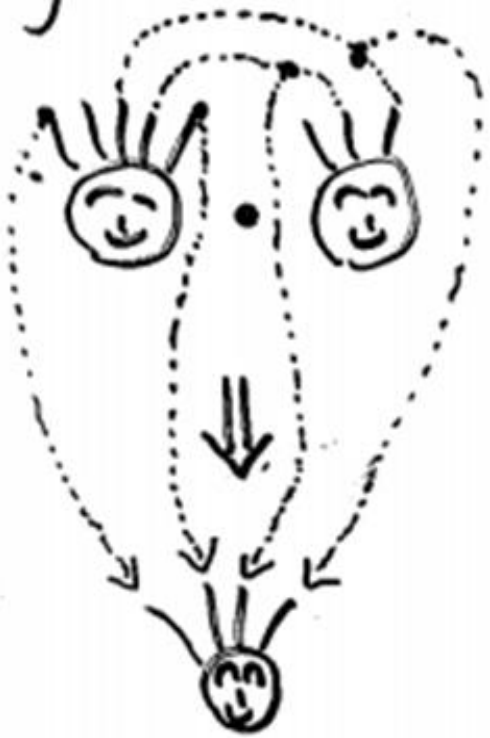
- (safety) Is this program/system in the safe (or correct, relevant, ...) classes of behaviours? Can the safety be preserved compositionally?

Functional Types



Process Types

- When it comes to processes, composition becomes:



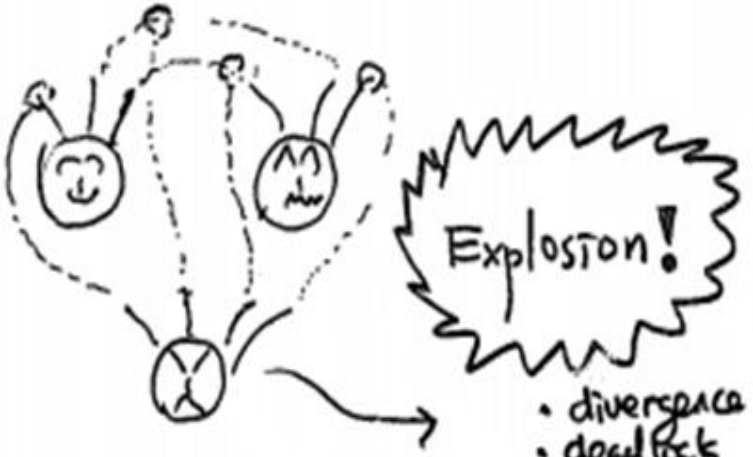
cf.

3 + 5

8

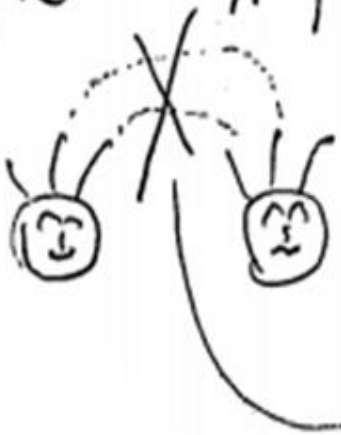
$P \bullet_\delta \delta \longmapsto_\tau \delta$

- But some composition is dangerous!



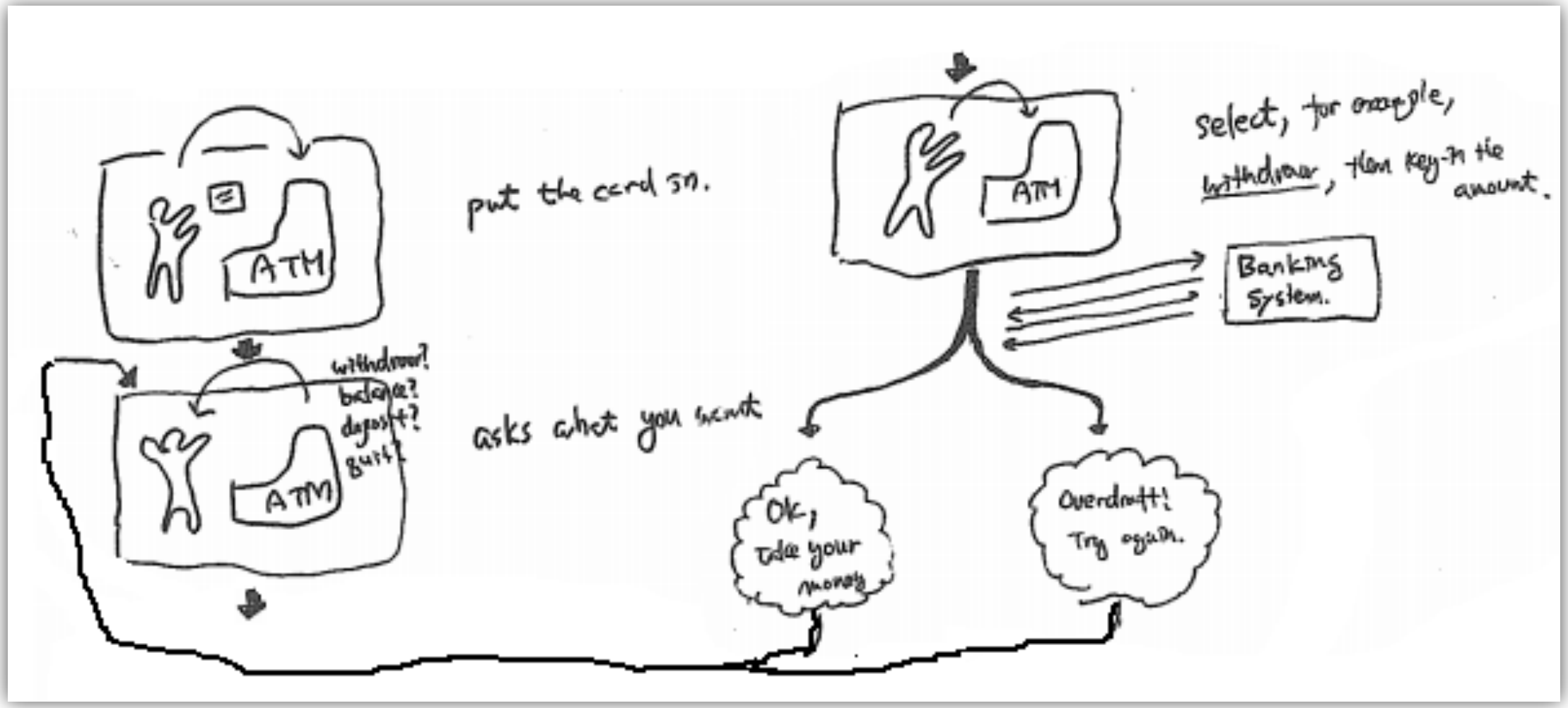
- divergence
- deadlock
- run-time error
- ⋮

- Therefore we type processes,



The connection is prohibited

Implementing ATM

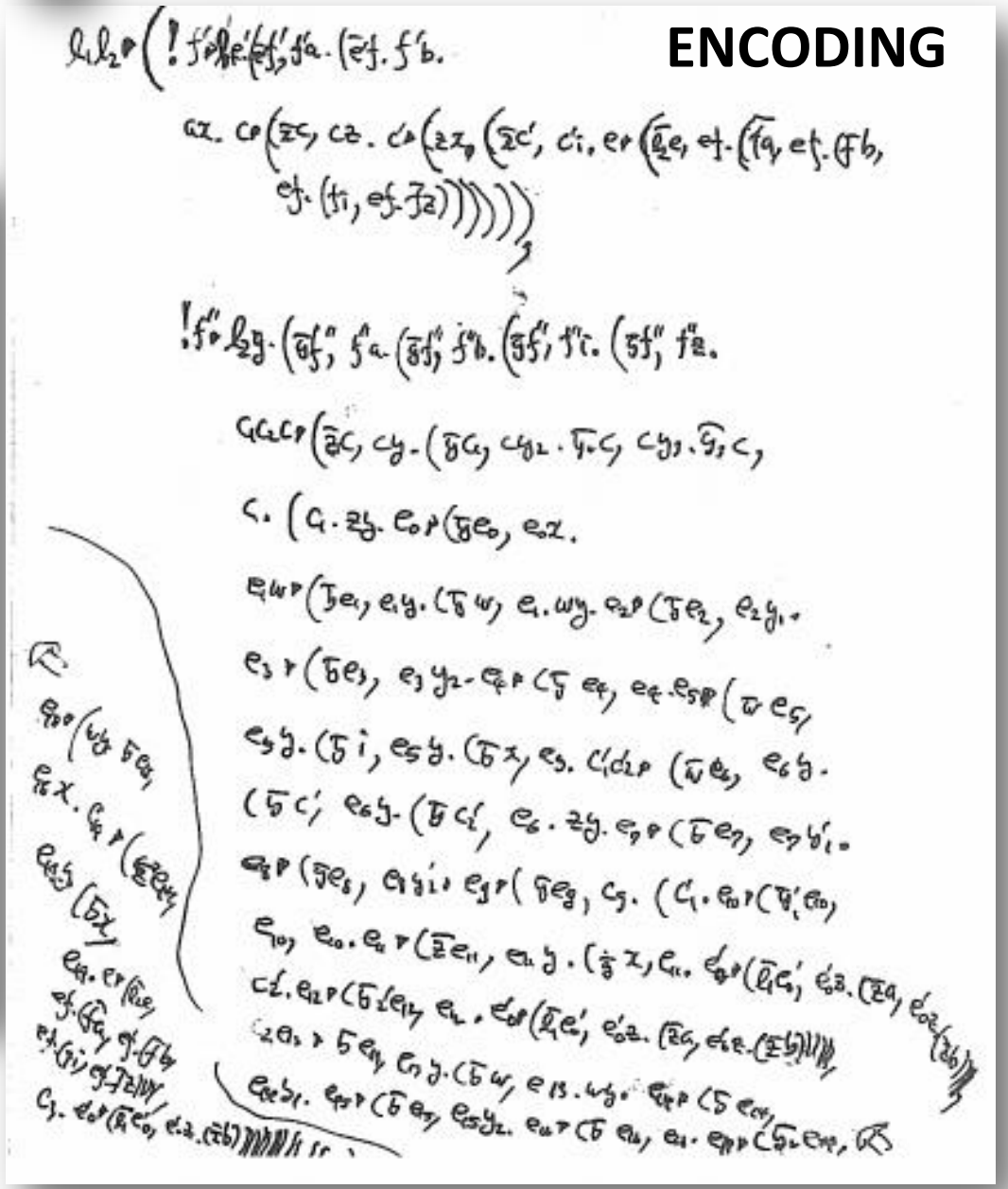


Implementing ATM

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

```
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



Dialogue between Industry and Academia

Binary Session Types [PARL'94, ESOP'98]

$\Downarrow$

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

$\Downarrow$

Formalisation of W3C WS-CDL [ESOP'07]

$\Downarrow$

Scribble at $\pi 4$ Technology

CDL Equivalent

- Basic example:

```
package HelloWorld {
    roleType YouRole, WorldRole;
    participantType You{YouRole}, World{WorldRole};
    relationshipType YouWorldRel between YouRole and WorldRole;
    channelType WorldChannelType with roleType WorldRole;

    choreography Main {
        WorldChannelType worldChannel;

        interaction operation=hello from=YouRole to=WorldRole
                relationship=YouWorldRel channel=worldChannel {
            request messageType=Hello;
        }
    }
}
```

Scribble Protocol

- *"Scribbling is necessary for architects, either physical or computing, since all great ideas of architectural construction come from that unconscious moment, when you do not realise what it is, when there is no concrete shape, only a whisper which is not a whisper, an image which is not an image, somehow it starts to urge you in your mind, in so small a voice but how persistent it is, at that point you start scribbling" - Kohei Honda 2007*

- Basic example:

```
protocol HelloWorld {
    role You, World;
    Hello from You to World;
}
```

Dialogue between Industry and Academia

Binary Session Types [PARL'94, ESOP'98]

⇓

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

⇓

Formalisation of W3C WS-CDL [ESOP'07]

⇓

Scribble at $\pi 4$ Technology

⇓

Multiparty Session Types [POPL'08]

⇓

Dialogue between Industry and Academia

Binary Session Types [PARL'94, ESOP'98]

⇓

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

⇓

Formalisation of W3C WS-CDL [ESOP'07]

⇓

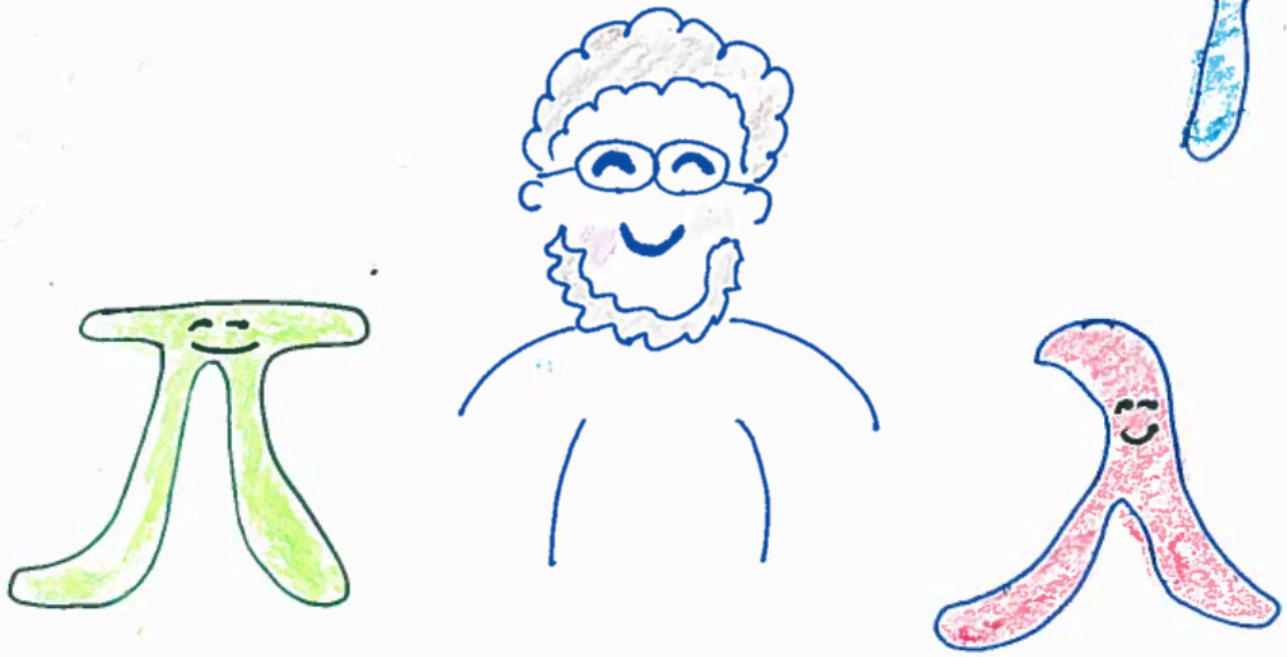
Scribble at $\pi 4$ Technology

⇓

Multiparty Session Types [POPL'08]

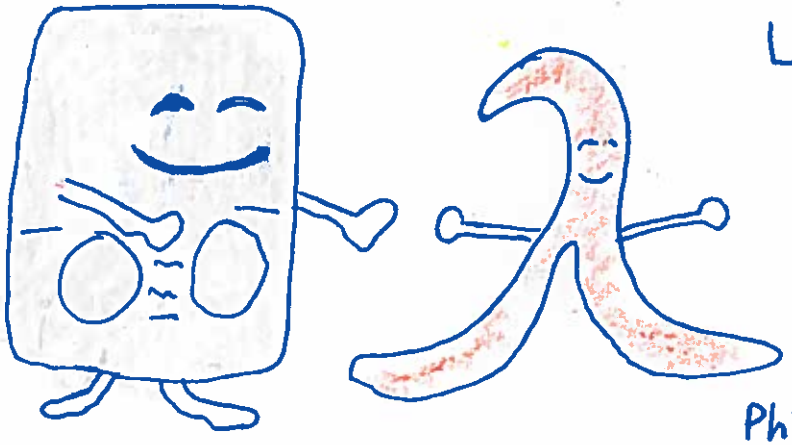
⇓

PHIL & SESSION TYPES



Imperial College London

199?

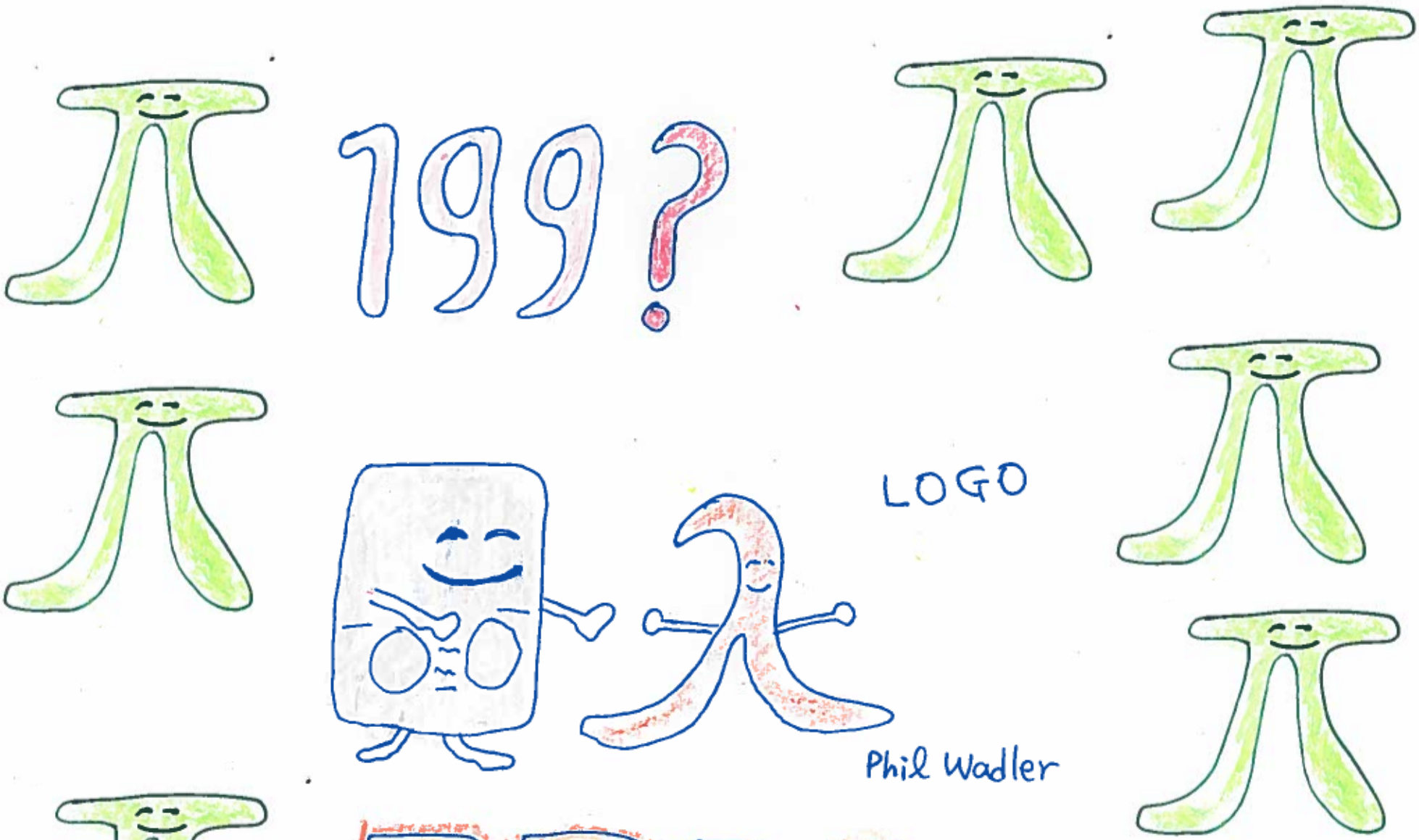


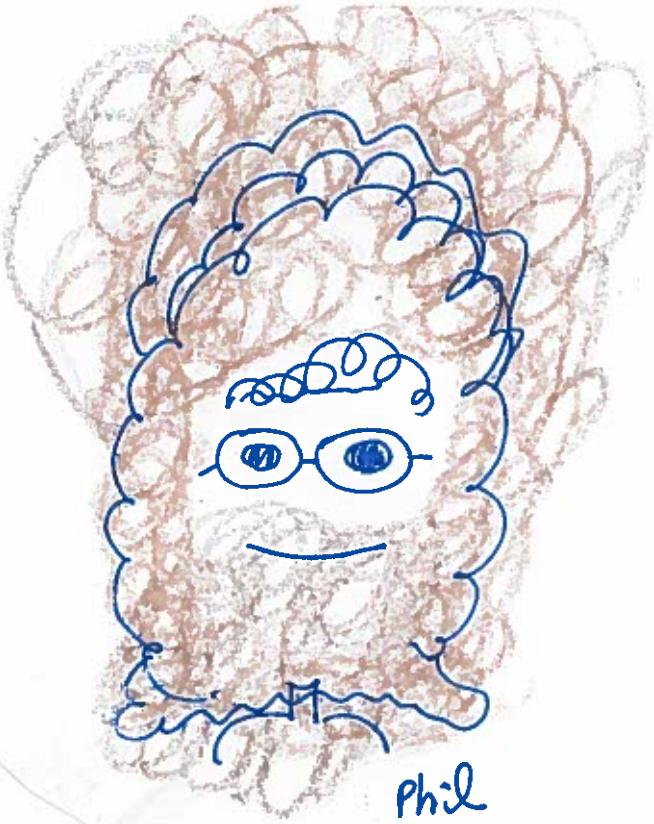
LOGO

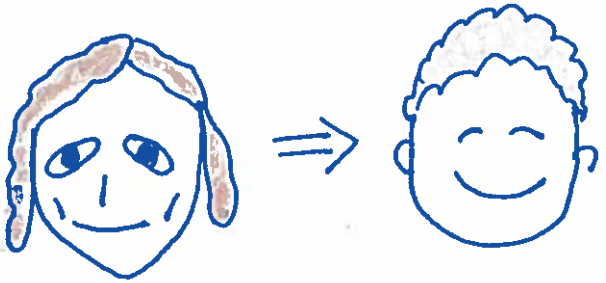
Phil Wadler

FPCA

Functional Programming Languages and Computer Architectures


199?
LOGO
Phil Wadler
FPCA
Functional
Programming
Languages and
computer
Architectures



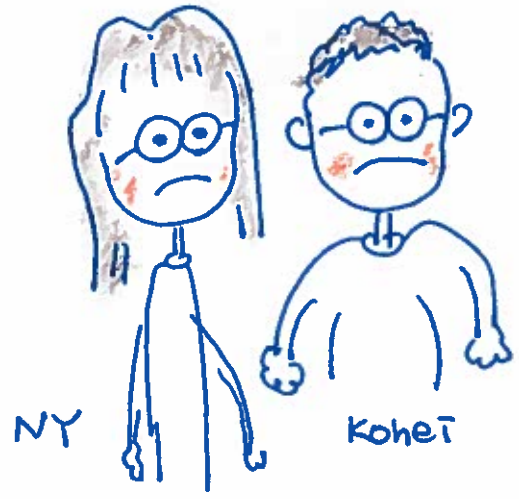


Jean-Jacque Levy

Andy Gordon

Luc Maranget

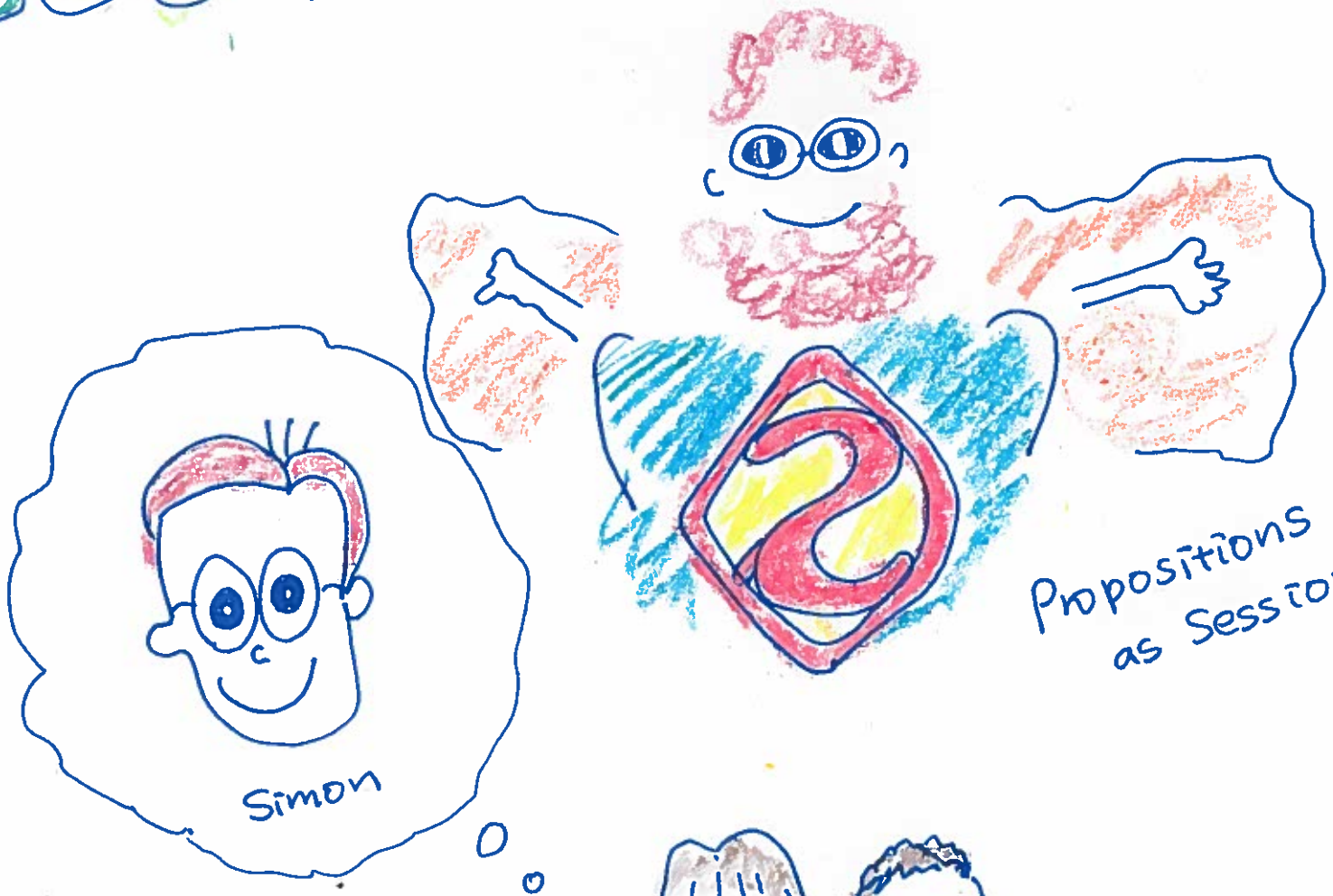
ABCD



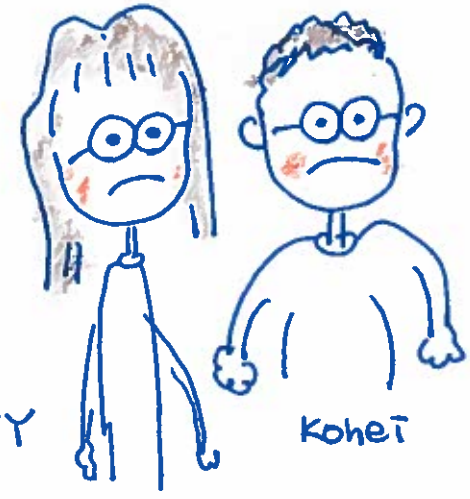
HOD

Imperial College

ABCD program grant 2013–2018 EPSRC



Propositions as Sessions @ ICFP'12



HOD

Imperial College