

# Session TYPES

AS A

DESCRIPTIVE



Nobuko  
Yoshida  
&  
Raymond Hu

25th  
Oct  
2017

TOOL

for DISTRIBUTED PROTOCOL

# 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



POPL 2008 Most Influential Paper Award

Kohei Honda, Nobuko Yoshida and Marco Carbone

Multiparty asynchronous session types





## 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.*

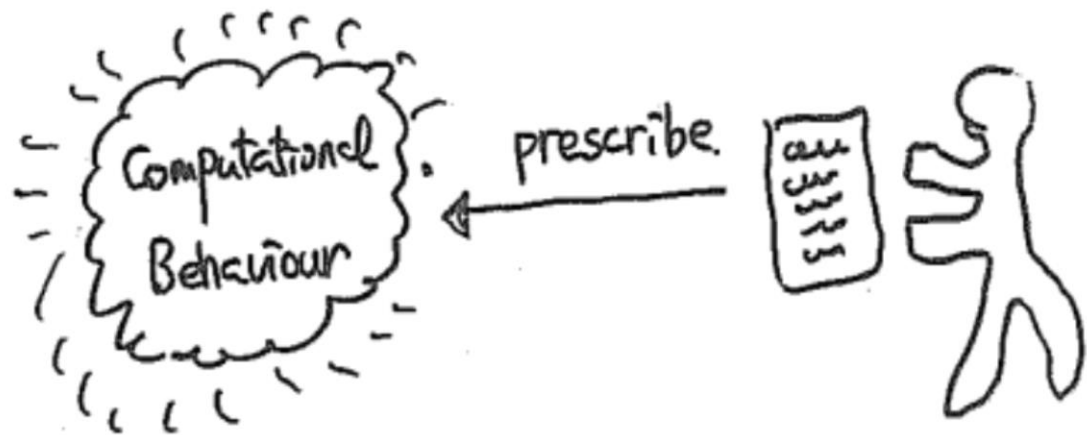
# Idioms for Interaction

— Invitation to Hacking in  $\pi$ -calculus —

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

- Imperative
- Functional
- Logical

- Programs : prescription of computational behaviours based on a certain abstraction.



# 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:  $\beta$ -reduction.

Logical: unification.

- Another element is how we can combine, on structure, these operations:

Imperative: sequential composition, if-then-else, while, procedures, module, ....

Functional: application, product, union, recursion, modules, .....

# UNSTRUCTURED:

```

data _stkl, 48 } stacks.
data _stkr, 48 }
data _i, 4 } index
data _j, 4 }
data _l, 4 } left/right limit
data _r, 4 }
data _x, 4 } pivot
data _w, 4 } temporary value.
data _s, 4 } stack pointer
data _a, 48 } table to be sorted.

mov $0, _s
mov $0, _stkl } 2nd instruction
mov $11, _stkr

L1: mov _s, rax } Top loop.
    mov _stkl(, rax, 4), rcx } l := stkl(s)
    mov rcx, _l
    mov _s, rax
    mov _stkr(, rax, 4), rcx } r := stkr(s)
    mov rcx, _r
    dec _s

L2: mov _l, rcx } j := l.
    mov rcx, _i
    mov _r, rcx } i := r.
    mov rcx, _j
    mov _l, rcx } rdx := l+r
    add _r, rcx
    mov rcx, rax } rdx := (l+r)/2
    mov rax, rdx
    shr $31, rdx
    add rdx, rax
    mov rax, rdx
    sar $1, rdx
    mov _a(, rdx, 4), rcx } x := a(rdx)
    mov rcx, _x

L3: mov _i, rax } third loop
    mov _a(, rax, 4), rdx } rdx := a(i)
    cmp rdx, _x
    jle L4
    inc _i
    jmp L3 } if rdx >= x goto (4)
    } else i := i+1
    } goto (3) (loop)

L4: mov _j, rax
    mov _a(, rax, 4), rdx } rdx := a(j)

```

# 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
    if i <= j then swap(i, j); i := i+1; j := j-1; end
  until i > j;
  if l < j then sort(l, j);
  if l < i then sort(i, r);
  end
  } Partition into two parts.
  } Recursively sort two parts.

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

```

# Quicksort in pure lambda:

$((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda q. \lambda l.$   
 $((\lambda x. x(\lambda xy. x))l)(\lambda x. x)$  } if l is not then nil.  
 $((\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda c. \lambda xy. x((\lambda x. x(\lambda py. x))x)y$  } concat.  
 $((\lambda xy. \lambda z. z(\lambda xy. y)xy)((\lambda x. x(\lambda xyz. y))x)(c((\lambda x. x(\lambda xyz. z))x)y$   
 $(q(\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda f. \lambda px. ((\lambda x. x(\lambda py. x))x)(\lambda x. x))$  } sort and  
 $(p((\lambda x. x(\lambda xyz. y))x))((\lambda xy. \lambda z. z(\lambda xy. y)xy)x$  Filter  
 $(f((\lambda x. x(\lambda xyz. z))x)))(f((\lambda x. x(\lambda xyz. z))x)))$   
 $(\lambda y. (((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda f'. \lambda xy. ((\lambda x. x(\lambda py. x))y$  } (λy. LTy. Car  
 $(\lambda xy. y)((\lambda x. x(\lambda py. x))x)(\lambda xy. y)(f'((\lambda x. x(\lambda py. y))x)((\lambda x. x(\lambda py. y)$   
 $y((\lambda x. x(\lambda xyz. y))l))((\lambda x. x(\lambda xyz. z))l)))$  } cdr l  
 $((\lambda xy. \lambda z. z(\lambda xy. y)xy)((\lambda x. x(\lambda xyz. y))l)$  } cons (car l)  
 $(q((\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda f. \lambda px. ((\lambda x. x(\lambda py. x))x)$  } sort and  
 $(\lambda x. x)(p((\lambda x. x(\lambda xyz. y))x))((\lambda xy. \lambda z. z(\lambda xy. y)xy)x$  Filter  
 $(f((\lambda x. x(\lambda xyz. z))x)))(f((\lambda x. x(\lambda xyz. z))x)))$   
 $(\lambda y. (((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda f''. \lambda xy. ((\lambda x. x(\lambda py. x))x$  } λy. Mz y  
 $((\lambda x. x(\lambda py. x))y)(\lambda xy. x)(\lambda xy. y))$  Car l  
 $((\lambda x. x(\lambda py. x))y)(\lambda xy. x)(f''((\lambda x. x(\lambda py. y))x$   
 $((\lambda x. x(\lambda py. y))y))y((\lambda x. x(\lambda xyz. y))l))((\lambda x. x(\lambda xyz. z))l))))$  } cdr l

# Quicksort with combinators:

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

$I = \lambda x. x$     $T = \lambda xy. x$     $F = \lambda xy. y$     $Y = (\lambda xy. y(xy))(\lambda xy. y(xy))$   
 $\text{cons} = \lambda xy. \lambda z. z F xy$     $\text{isnil} = \lambda x. x T$   
 $\text{car} = \lambda x. x (\lambda yz. y)$     $\text{cdr} = \lambda x. x (\lambda yz. z)$   
 $\text{concat} = Y(\lambda c. \lambda xy. x (\text{isnil } x) y (\text{cons } (\text{car } x) (c (\text{cdr } x)) y)$   
 $\text{filter} = Y(\lambda f. \lambda px. (\text{isnil } x) I (p (\text{car } x)) (\text{cons } x (\text{filter } (\text{cdr } x))))$   
 $\text{iszero} = \lambda x. x T$     $\text{pred} = \lambda x. x F$     $(\text{filter } (\text{cdr } x))$   
 $\text{LT} = Y(\lambda f. \lambda xy. (\text{iszero } y) F ((\text{iszero } x) F (f (\text{pred } x) (\text{pred } y))$   
 $\text{ME} = Y(\lambda f. \lambda xy. (\text{iszero } x) ((\text{iszero } y) I F) ((\text{iszero } y) (T) y))$   
 $(f (\text{pred } x) (\text{pred } y))$

## 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 $\pi$ -calculus as a Descriptive Tool

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

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

$$\text{with } \pi ::= x(\bar{y}) \mid \bar{x}(y).$$

$\lambda$  in  $\pi$

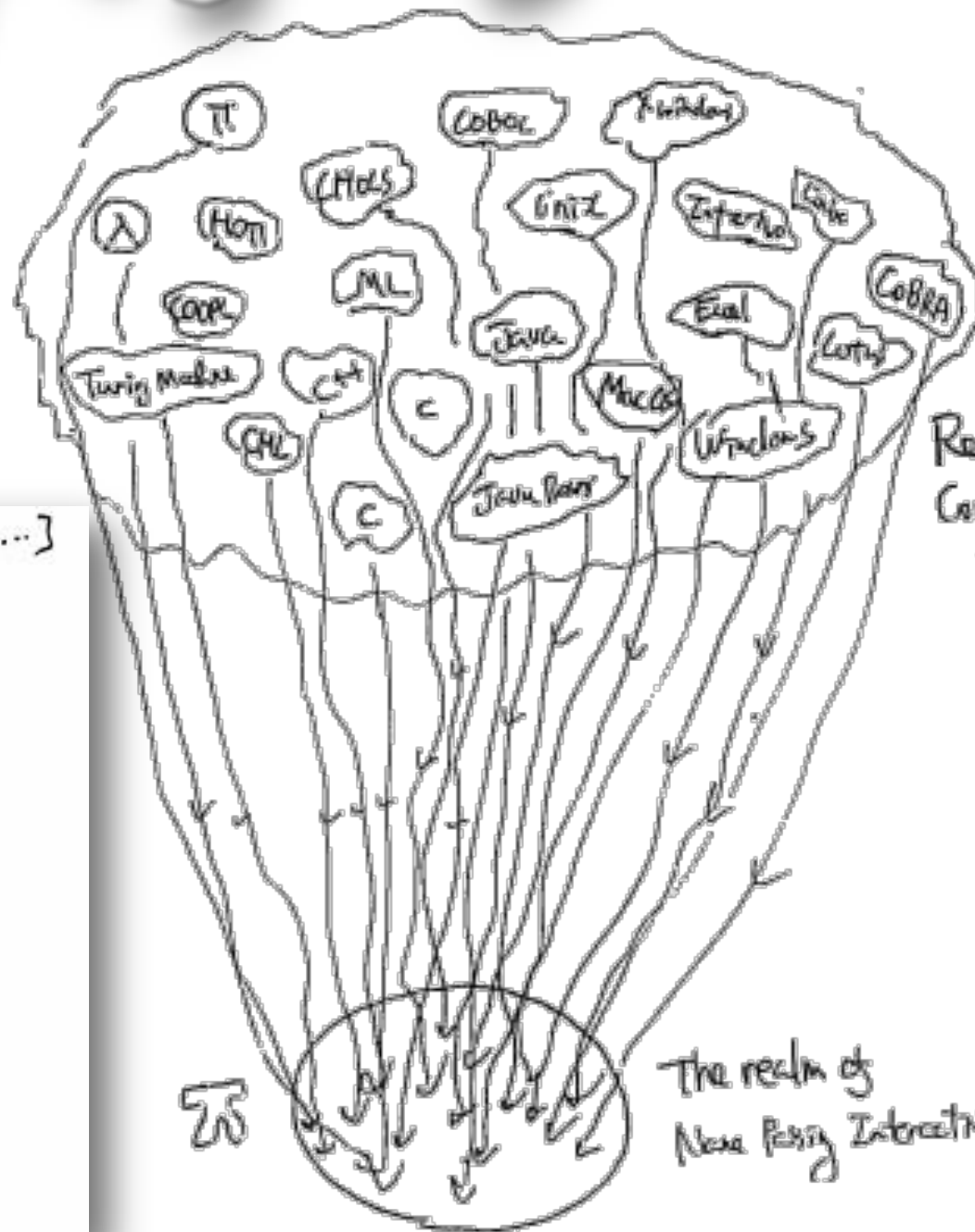
$$[x]_u \stackrel{\text{def}}{=} \bar{x}(u).$$

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

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

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

# \* Examples of Representable Computation.



- $\lambda$ -calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- $\omega$ -order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellare and Scott 93]
- Abstract "constant" interaction [HRS4]
- Strategies on Frames [HO95]

Realms of Computational Behaviour

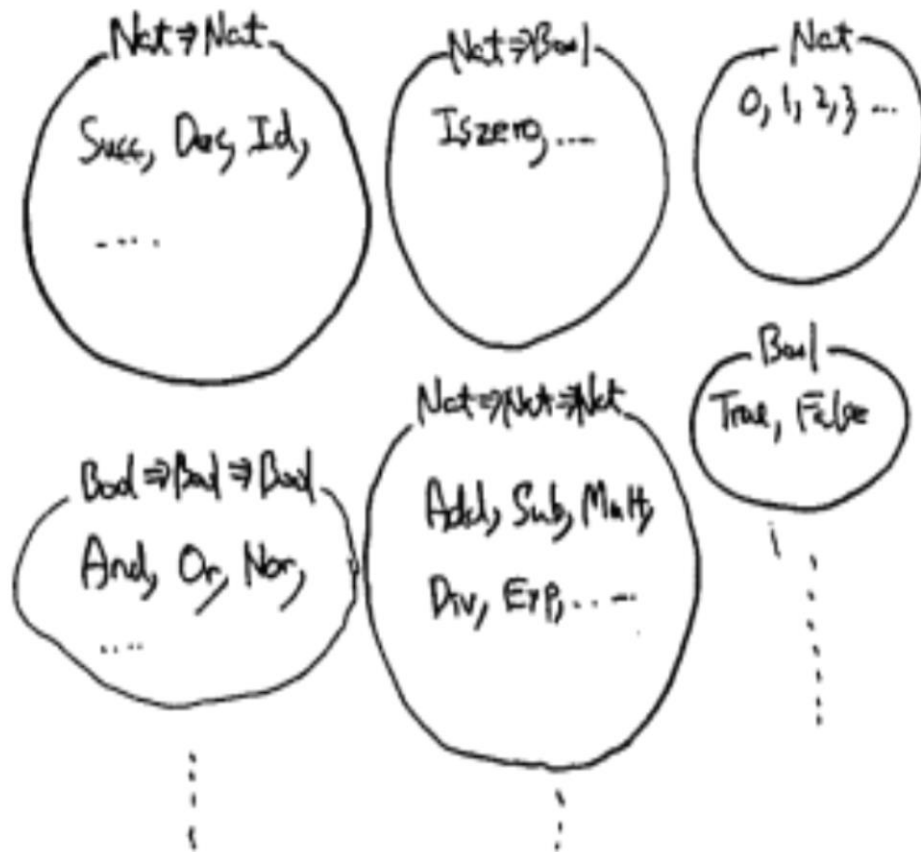
The realm of New Programming Interactions

# The Role of Types in $\pi$ -Calculus.

- (classification) How can we classify name-passing interactive behaviours, i.e. behaviours representable in  $\pi$ -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



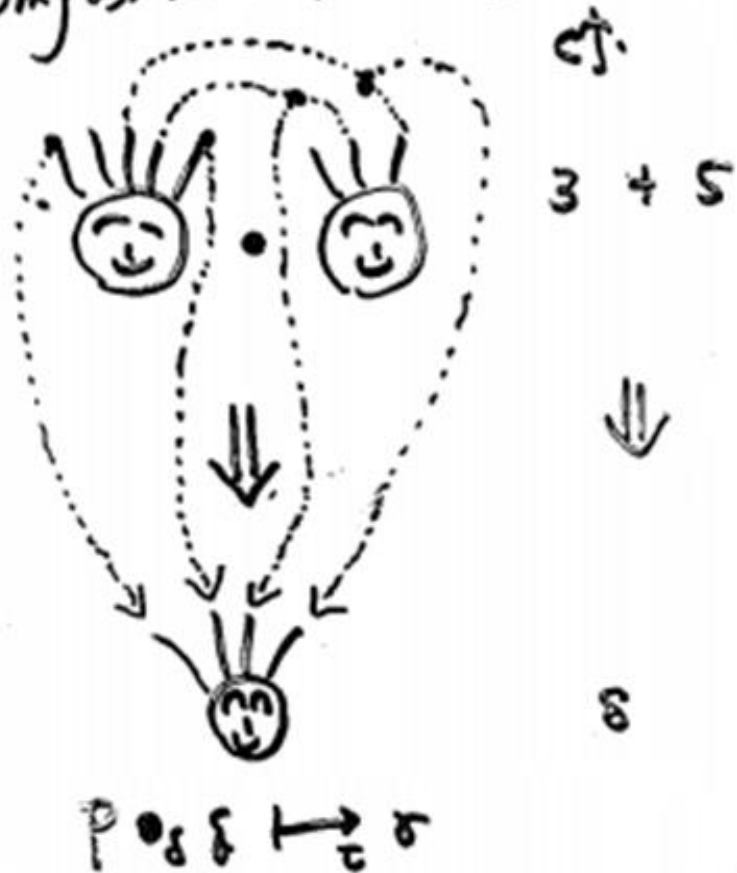
with operation!

$$\left\{ \begin{array}{l} f : \alpha \Rightarrow \beta \bullet e : \alpha = f \bullet e : \beta. \\ \text{else undefined.} \end{array} \right.$$

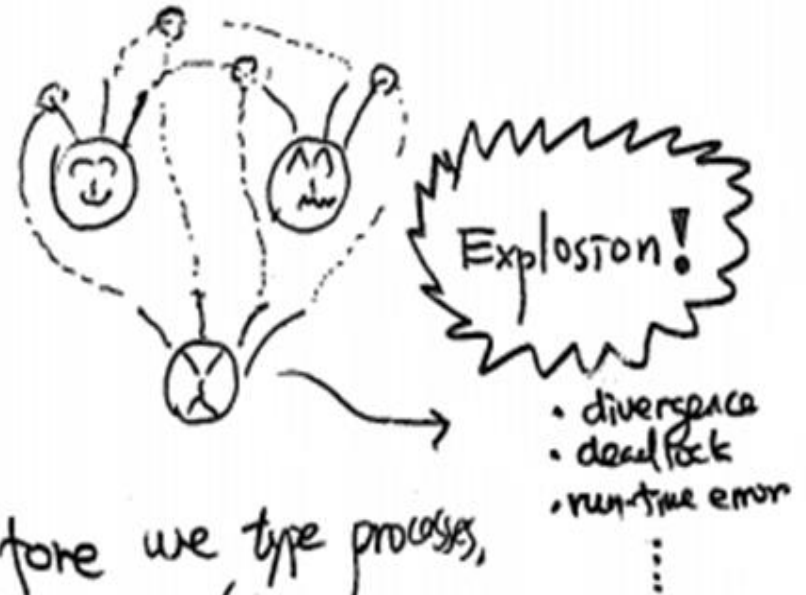
function application.

# Process Types

- When it comes to processes, composition becomes:



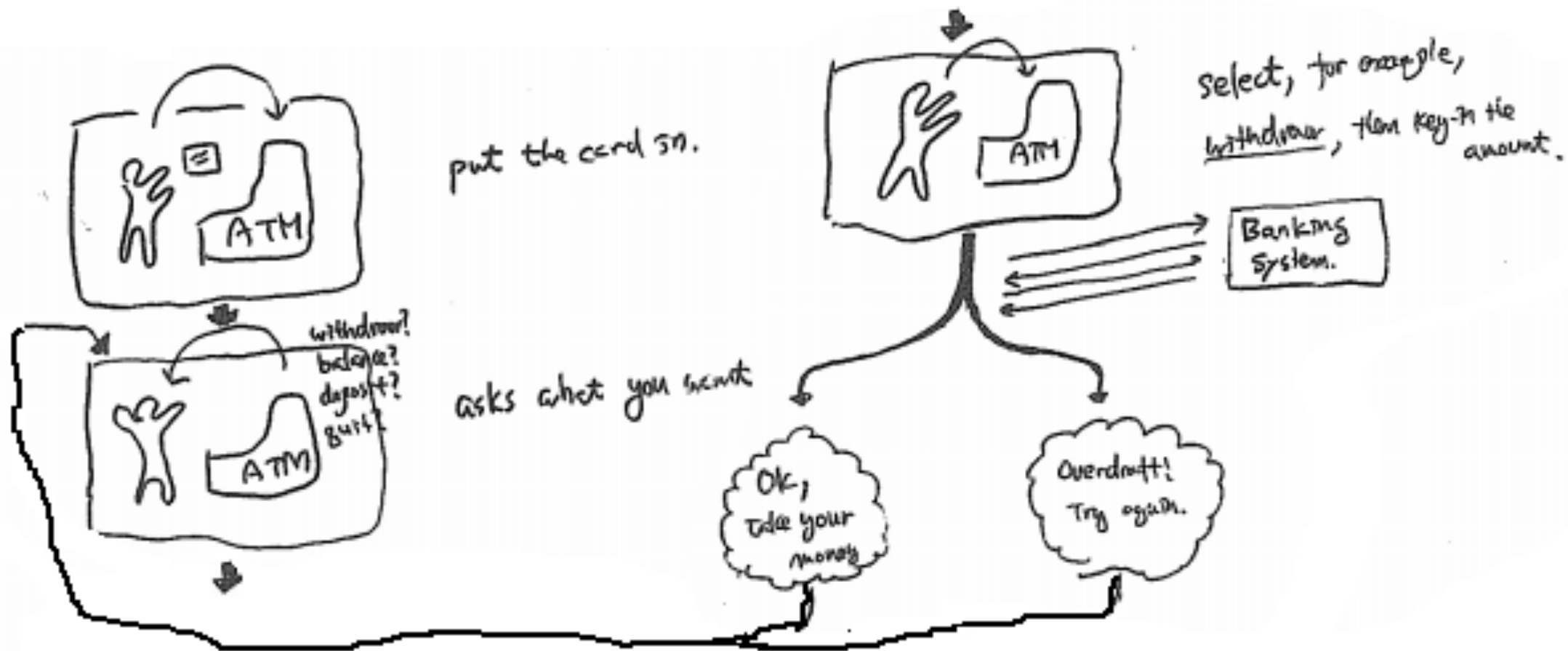
- But some composition is dangerous!



- Therefore we type processes,



# Implementing ATM



# Implementing ATM

$ATM^{cb} =$

$z < z :: z ? ID ; [ ? w d ; ? X ;$	$z \leftrightarrow user.$
$\bar{b} < w :: w ! w t ; ! ID ; ! X ;$	$w \leftrightarrow bank.$
$[ ? ok ;$	$x :$
$z ! ok ; ! X ; ATM^{cb} ;$	$z \leftrightarrow user$
$? overdraft :$	$w \leftrightarrow bank$
$z ! over ; ATM^{cb} ] ;$	$x \leftrightarrow user.$
$? bal ;$	$z \leftrightarrow user$
$\bar{b} < w :: w ! bal ; ? X ;$	$w \leftrightarrow bank$
$z ! X ; ATM^{cb} ]$	

## ENCODING

$z \leftrightarrow user.$

$z < z :: z ? ID ; [ ? w d ; ? X ;$   
 $\bar{b} < w :: w ! w t ; ! ID ; ! X ;$   
 $[ ? ok ;$   
 $z ! ok ; ! X ; ATM^{cb} ;$   
 $? overdraft :$   
 $z ! over ; ATM^{cb} ] ;$   
 $? bal ;$   
 $\bar{b} < w :: w ! bal ; ? X ;$   
 $z ! X ; ATM^{cb} ]$

$z \leftrightarrow user.$

$w \leftrightarrow bank.$

$x :$

$z \leftrightarrow user$

$w \leftrightarrow bank$

$x \leftrightarrow user.$

$z \leftrightarrow user$

$w \leftrightarrow bank$

*(Handwritten notes and diagrams on the right page, including arrows and additional text, are mostly illegible due to blurriness and handwriting.)*

# 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

# 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



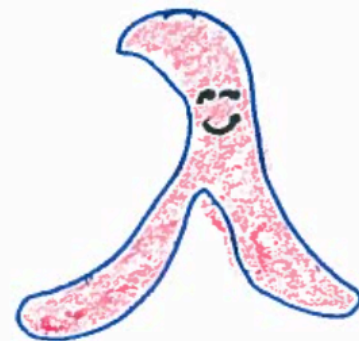
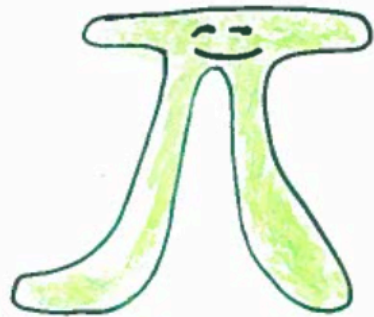
Multiparty Session Types [POPL'08]



PHIL &

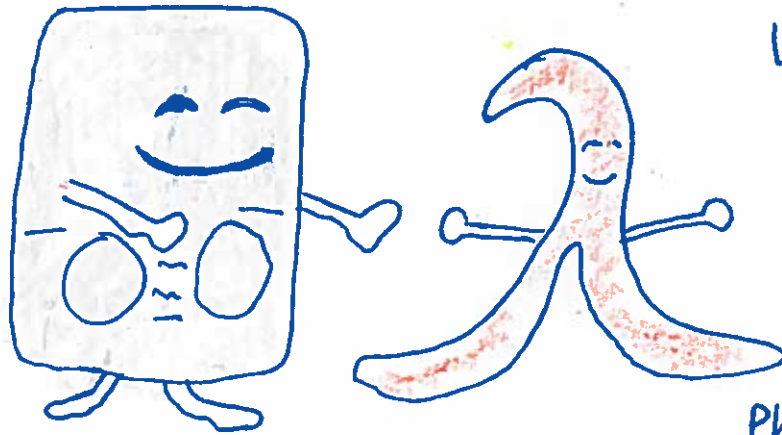
SESSION

TYPES



Imperial  
College  
London

1999?

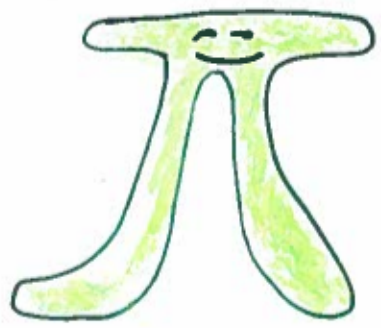


LOGO

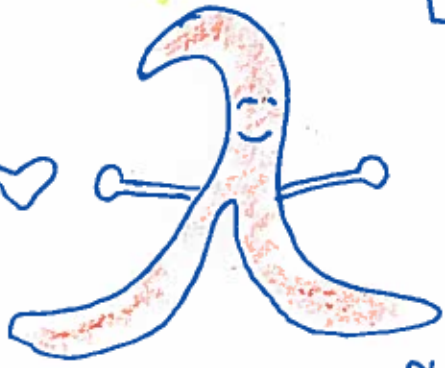
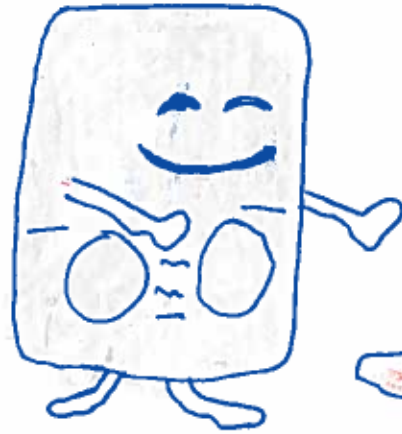
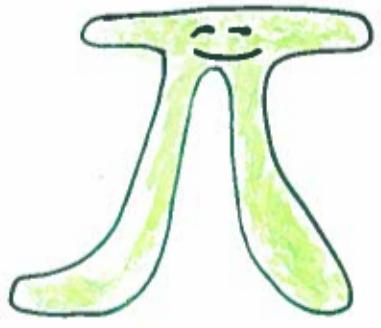
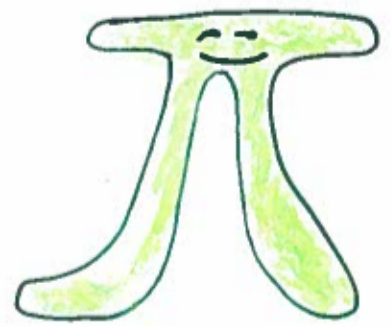
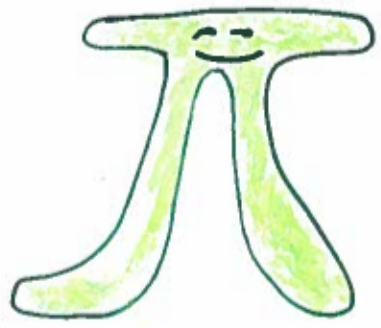
Phil Wadler

FPCA

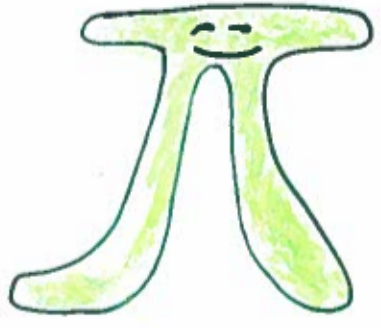
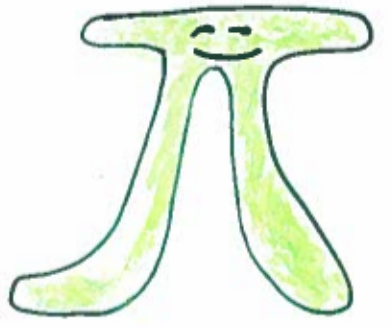
Functional  
Programming  
Languages and  
Computer  
Architectures



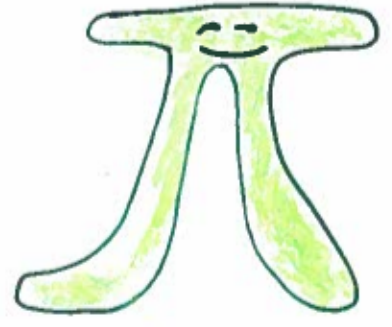
1999?



LOGO

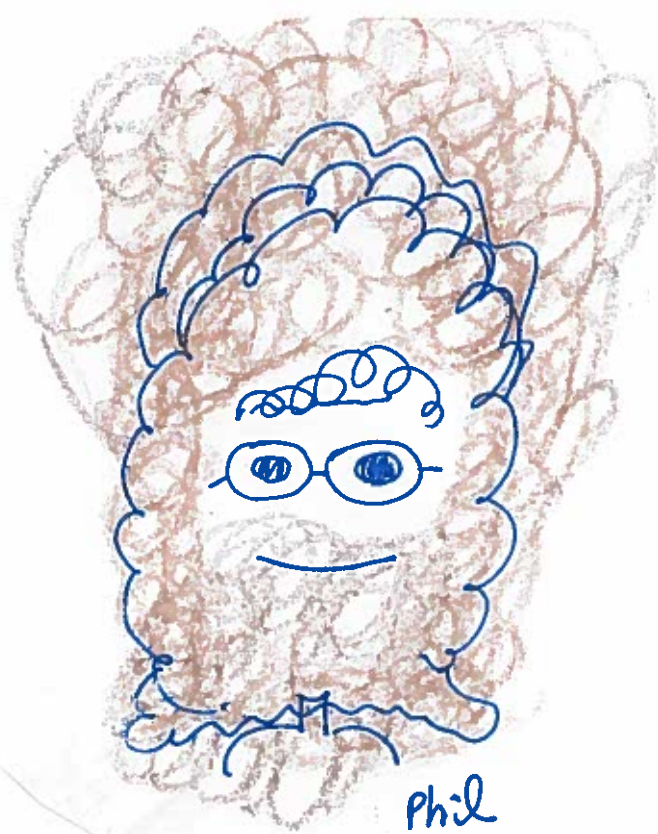


Phil Wadler

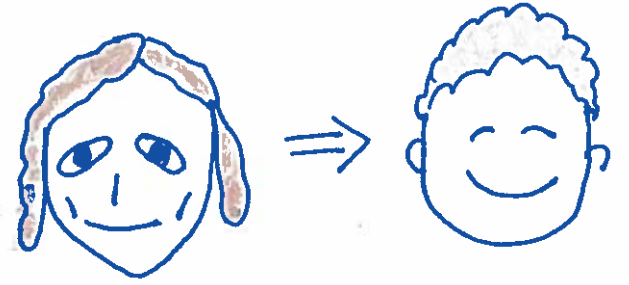


FPCA

Functional Programming Languages and Computer Architectures



Phil



Jean - Jacques Levy

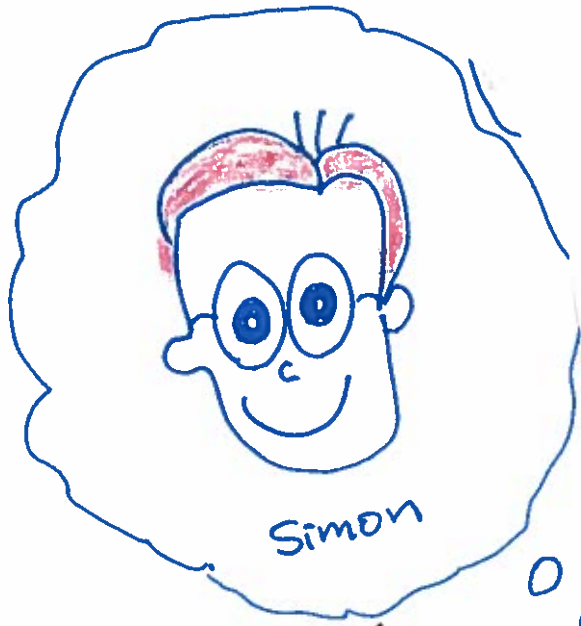


Andy Gordon

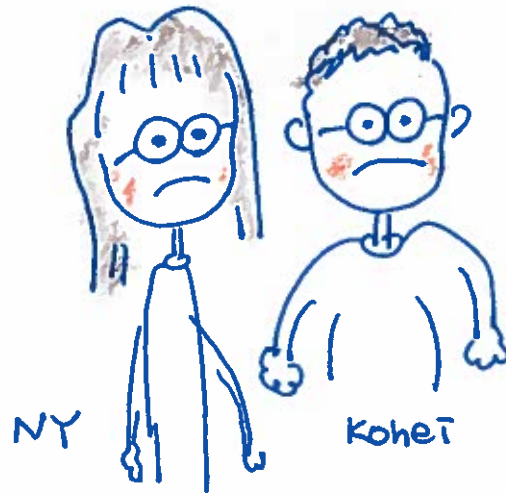


Luc Maranget

ABCD



0  
0  
6



SUSAN E

HOD

Imperial College

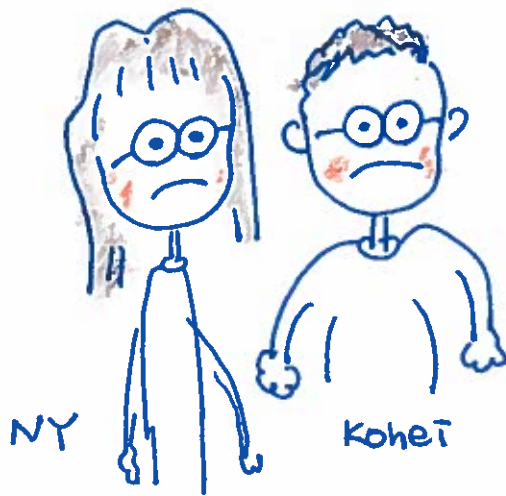
ABCD

program grant 2013-2018

EPSRC



Propositions  
as Sessions @ ICFP'12



HOD

Imperial College