on Polymorphic Sessions and Functions or a Tale of Two Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

http://mrg.doc.ic.ac.uk

Mobility Research Group

session type

π-calculus, Session Types research at Imperial College

Home | People | Publications | Grants | Talks | Tutorials | Tools | Awards | Kohei Honda

NEWS

The paper *Multiparty asynchronous session types* by Kohei Honda, Nobuko Yoshida, and Marco Carbone, published in POPL 2008 has been awarded the ACM SIGPLAN Most Influential POPL Paper Award today at POPL 2018.

» more

10 Jan 2018

Estafet has published a page on their usage of the Scribble language developed in our group with RedHat and other industry partners.

» more

25 Sep 2017

Nick spoke at Golang UK 2017 on applying behavioural types to verify concurrent Go programs.

SELECTED PUBLICATIONS

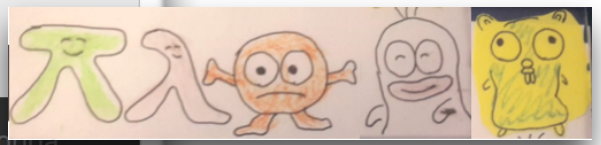
2018

Julien Lange , Nicholas Ng , Bernardo Toninho , Nobuko Yoshida : A Static Verification Framework for Message Passing in Go using Behavioural Types. *To appear in* ICSE 2018 .

Bernardo Toninho , Nobuko Yoshida : Depending On Session Typed Process. *To appear in* FoSSaCS 2018 .

Bernardo Toninho , Nobuko Yoshida : On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings. *To appear in* ESOP 2018 .

Rumyana Neykova , Raymond Hu , Nobuko Yoshida , Fahd Abdeljallal : Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#. *To appear in* CC 2018 .



Post-docs:

Simon CASTELLAN

David CASTRO

Francisco FERREIRA

Raymond HU

Rumyana NEYKOVA

Nicholas NG

Alceste SCALAS

PhD Students:

Assel ALTAYEVA

Juliana FRANCO

Eva GRAVERSEN

POPL 2008 MOST INFLUENTIAL PAPER AWARD

LICS 2018 TEST OF TIME AWARD

LICS'98

A fully abstract game semantics for general references

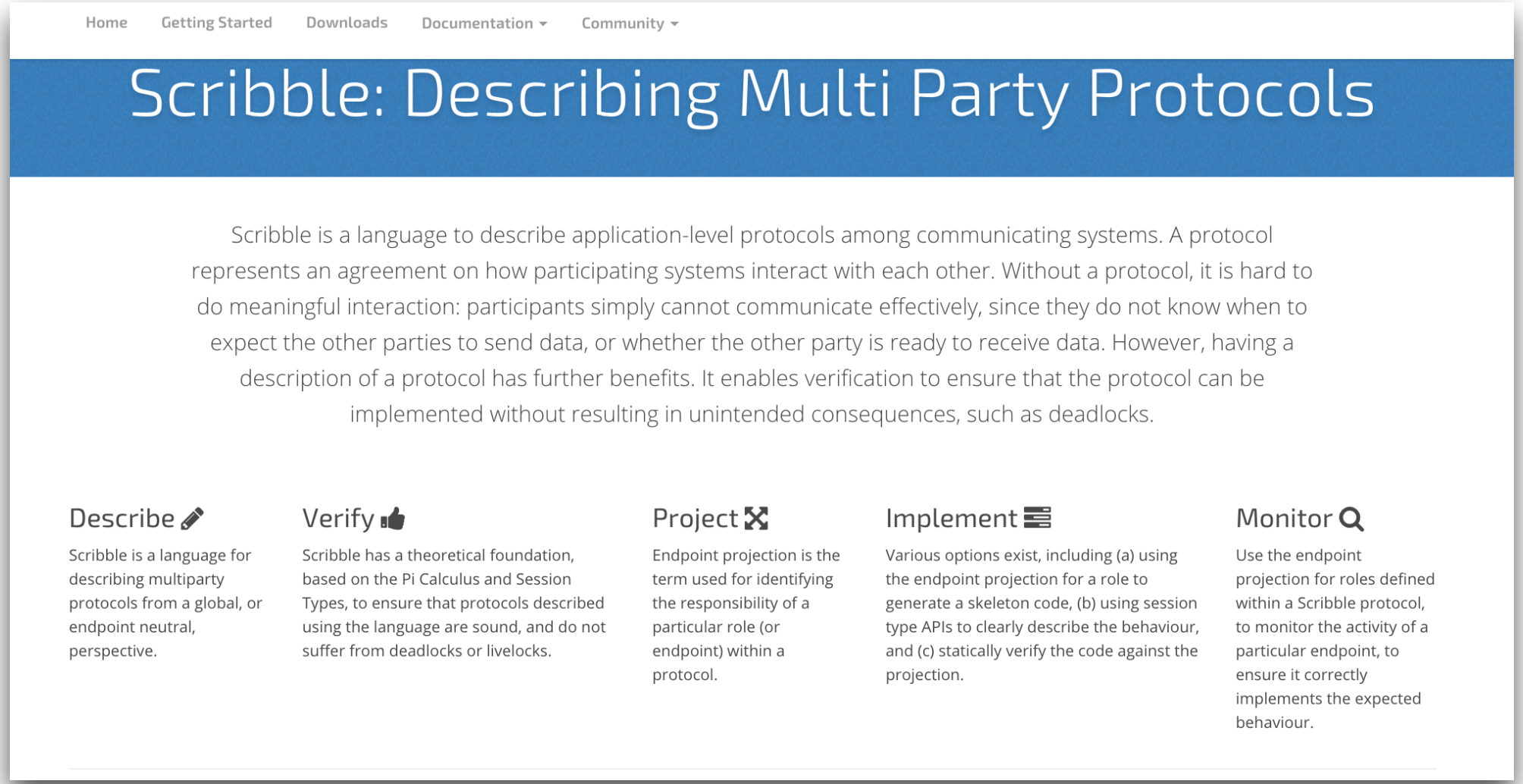
Samson Abramsky Kohei Honda
LFCS, University of Edinburgh
{samson,kohei}@dcs.ed.ac.uk

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.

www.scribble.org

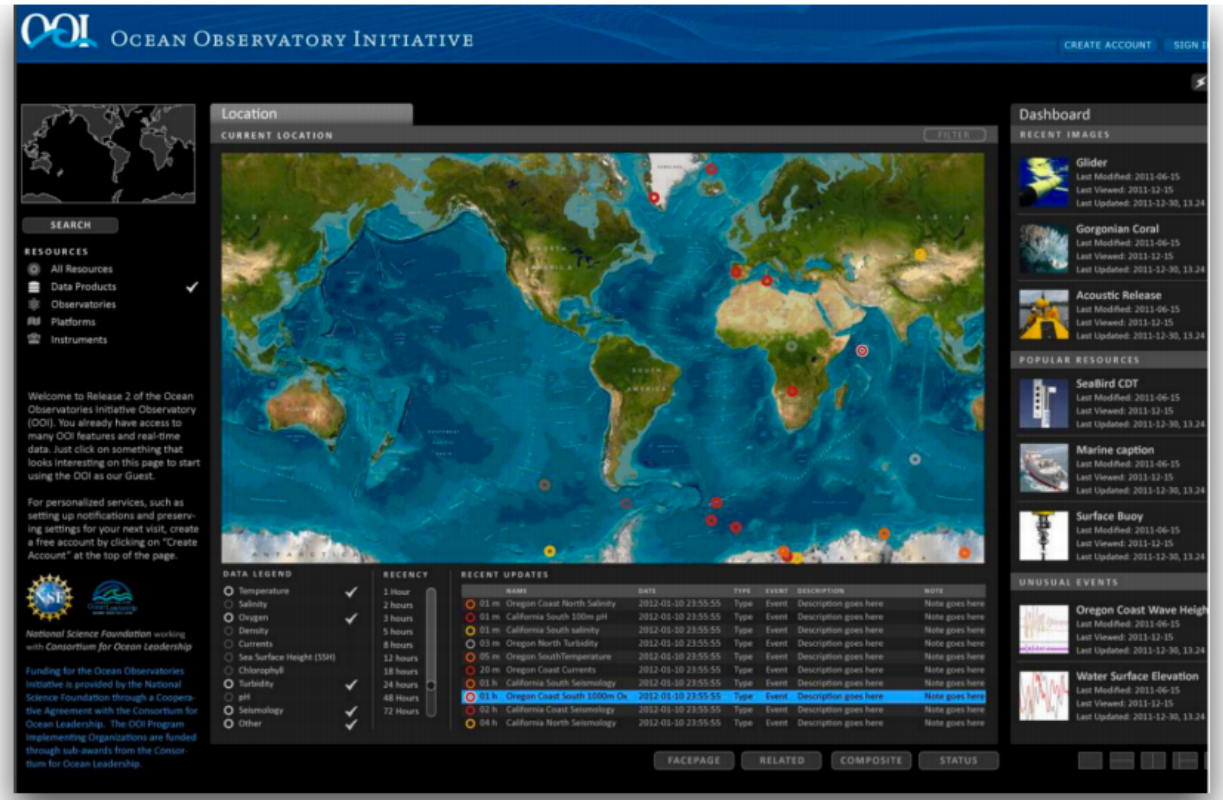


Online tool : http://scribble.doc.ic.ac.uk/

```
1  module examples;
2
3  global protocol HelloWorld(role Me, role World) {
4    hello() from Me to World;
5    choice at World {
6      goodMorning1() from World to Me;
7    } or {
8      goodMorning1() from World to Me;
9    }
10 }
11
```

Load a sample | Check | Protocol: examples.HelloWorld | Role: Me | Project | Generate Graph

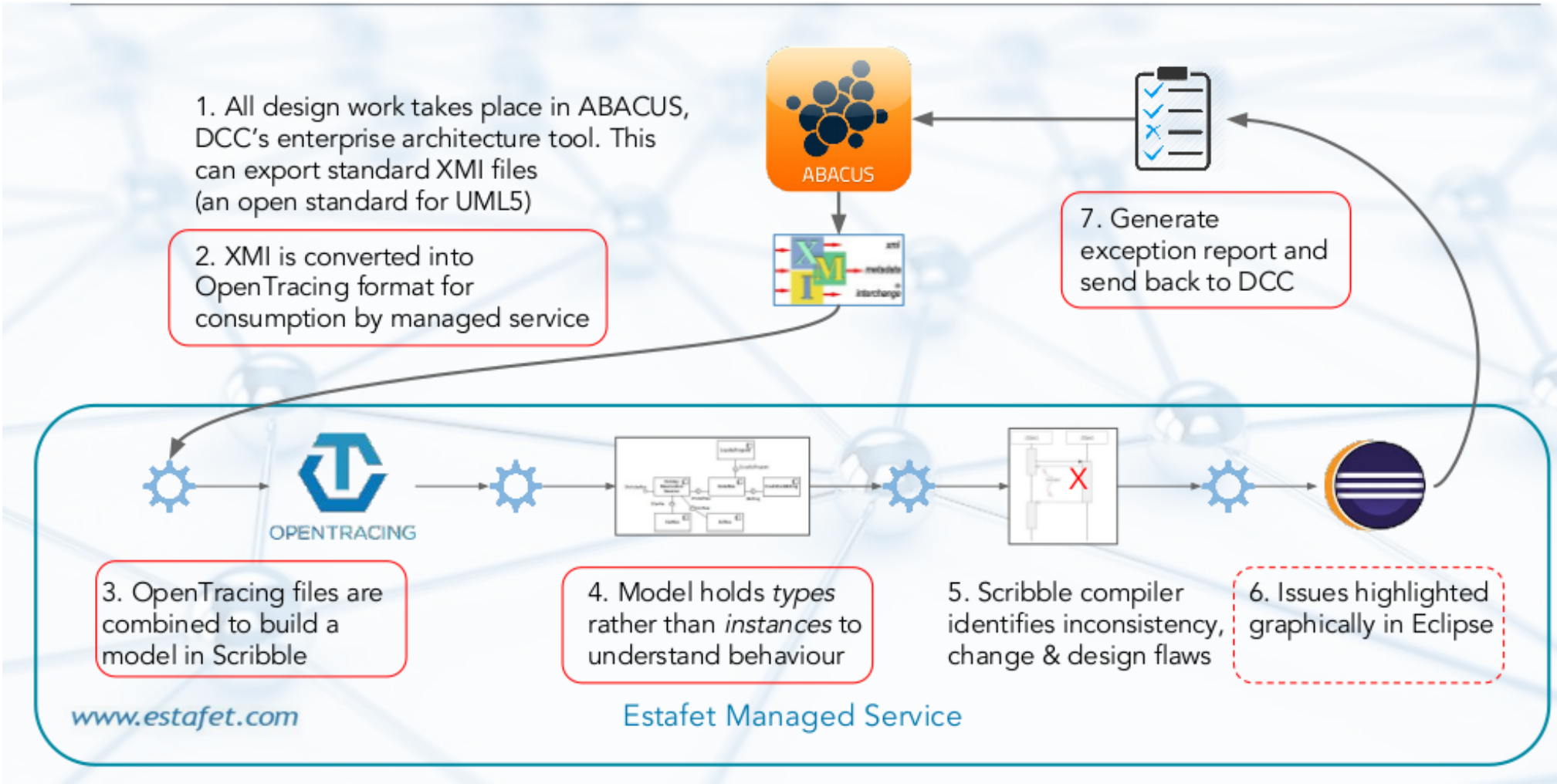
OOI Collaboration



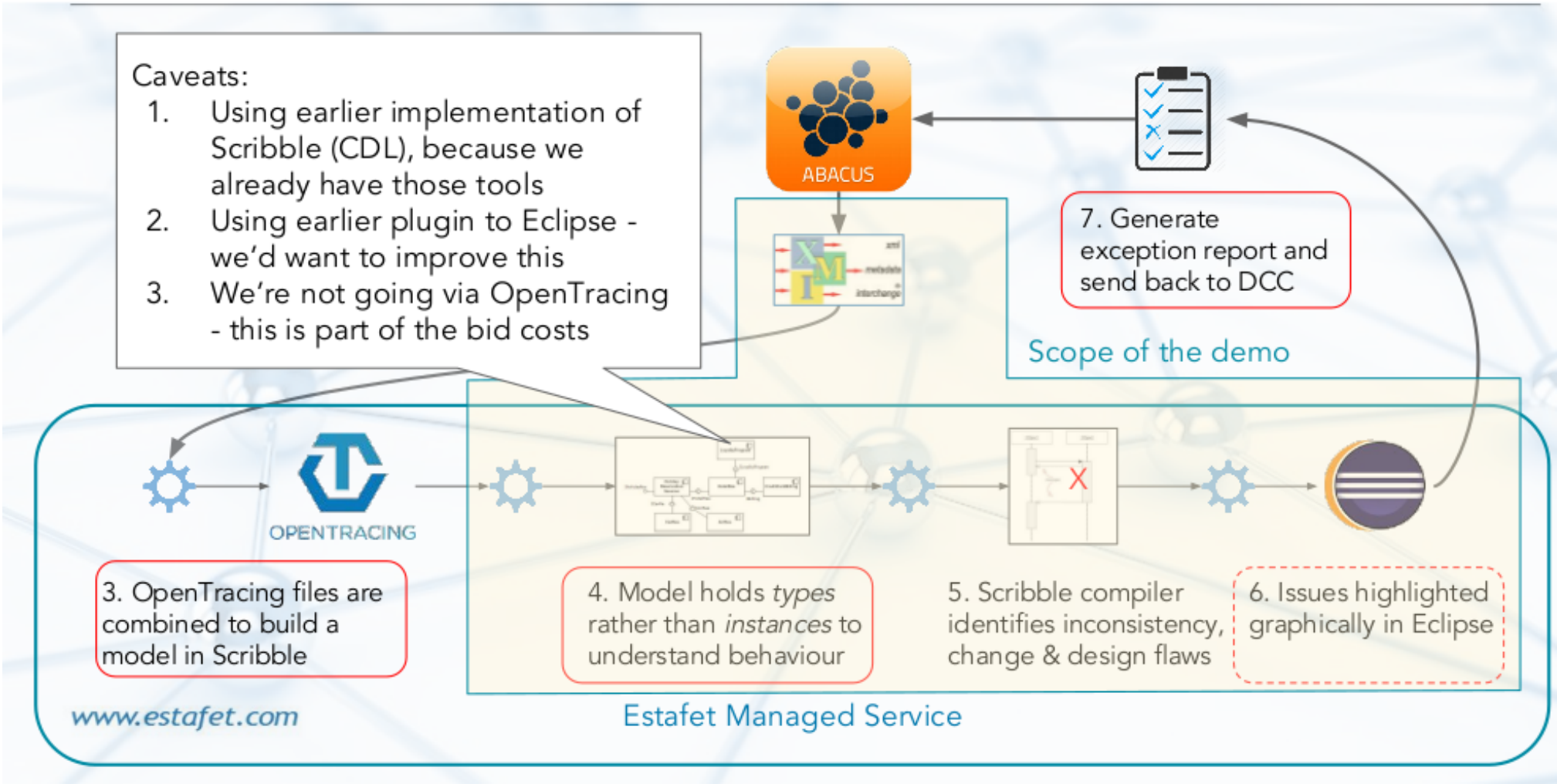
- **TCS'16:** Monitoring Networks through Multiparty Session Types. Laura Bocchi , Tzu-Chun Chen , Romain Demangeon , Kohei Honda , Nobuko Yoshida
- **LMCS'16**: Multiparty Session Actors. Rumyana Neykova, Nobuko Yoshida
- **FMSD'15:** Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon , Kohei Honda , Raymond Hu , Rumyana Neykova , Nobuko Yoshida
- **TGC'13:** The Scribble Protocol Language. Nobuko Yoshida , Raymond Hu , Rumyana Neykova , Nicholas Ng

End-to-End Switching Programme by DCC

Estafet
Innovate | Deliver | Transform



End-to-End Switching Programme by DCC



CC'18

A Session Type Provider

Compile-Time API Generation of Distributed Protocols with Refinements in F#

Rumyana Neykova
Imperial College London
United Kingdom

Raymond Hu
Imperial College London
United Kingdom

Nobuko Yoshida
Imperial College London
United Kingdom

Fahd Abdeljallal
Imperial College London
United Kingdom

Abstract

We present a library for the specification and implementation of distributed protocols in native F# (and other .NET languages) based on multiparty session types (MPST). There are two main contributions. Our library is the first practical development of MPST to support what we refer to as *interaction refinements*: a collection of features related to the refinement of *protocols*, such as message-type refinements (value constraints) and message-value dependent control flow. A well-typed endpoint program using our library is guaranteed to perform only compliant session I/O actions conforming to the refined protocol, up to premature termination. Moreover, our library is developed as a session *type provider*,

1 Introduction

Type providers [20, 27] are a .NET feature for a form of compile-time meta programming, designed to bridge between programming in statically typed languages such as F# and C#, and working with so-called *information spaces*—structured data sources such as SQL databases or XML data.

A type provider works as a compiler plugin that performs on-demand generation of *types*: it takes a schema for an external information space, and generates types that allow the data to be manipulated via a strongly-typed interface, with benefits such as static error detection and IDE auto-completion. For example, an instantiation of the in-built type provider for WSDL Web services [6] may look like

shots fired @zeeshanlakhani · Mar 12

Replying to @graydon_pub @dsyme

Awesome!

Brendan Zabarauskas @brendanzab ·

Replying to @graydon_pub

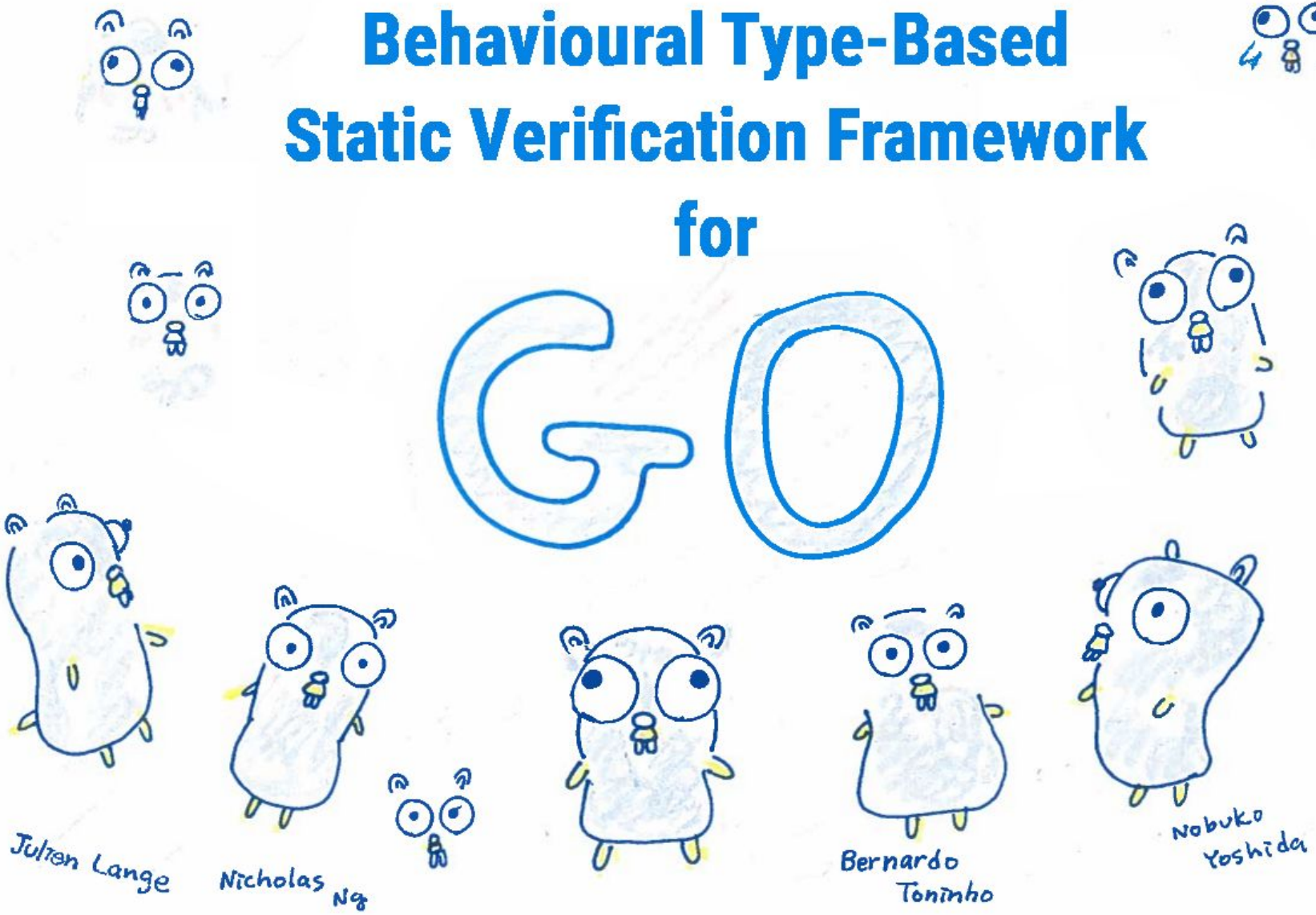
This stuff fills me with hope!

Ryan Riley @panesofglass · Mar 12

Replying to @graydon_pub

This is amazing! I guess I need to switch

Behavioural Type-Based Static Verification Framework for Go



Imperial College London

News

Home | College and Campus | Science | Engineering | Health | Business | Search here... | Go

Go concurrency verification research at DoC grabs headline

POPL'17

currency
erates a
tured in the
which
nteresting
easily
e of the
L (Principles

ICSE'18

the morning paper

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

Home About InfoQ QR Editions Subscribe

A static verification framework for message passing in Go using behavioural types

JANUARY 25, 2018

tags: Concurrency, Programming Languages

A static verification framework for message passing in Go using behavioural types Lange et al., *ICSE 18*

With thanks to Alexis Richardson who first forwarded this paper to me.

We're jumping ahead to ICSE 18 now, and a paper that has been accepted for publication there later this year. It fits with the theme we've been exploring this week though, so I thought I'd cover it now. We've seen verification techniques applied in the context of **Rust** and **JavaScript**, looked at the integration of **linear types in Haskell**, and today it is the turn of Go!

SUBSCRIBE

never miss an issue! The Morning Paper delivered straight to your inbox.

SEARCH

type and press enter

ARCHIVES

Select Month

MOST READ IN THE LAST FEW DAYS

Selected Publications 2017/2018

- **[LICS'18]** Romain Demangeon, NY: Casual Computational Complexity of Distributed Processes.
- **[CC'18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS'18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP'18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP'18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE'18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types
- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming..
- **[COORDINATION'17]** Keigo Imai, NY, Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS'17]** Julien Lange, NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu, NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Rumyana Neykova, NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL'17]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY: Fencing off Go: Liveness and Safety for Channel-based Programming.

Selected Publications 2017/2018

- **[LICS'18]** Romain Demangeon, NY: Casual Computational Complexity of Distributed Processes.
- **[CC'18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS'18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP'18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP'18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE'18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types.
- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- **[COORDINATION'17]** Keigo Imai, NY, Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS'17]** Julien Lange, NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu, NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Rumyana Neykova, NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL'17]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY: Fencing off Go: Liveness and Safety for Channel-based Programming.

POPL 2019 Research Papers

Write a Blog >>

15:21 - 16:27: Research Papers - Capabilities and Session Types I at POPL Track 2

15:21 - 15:43 *Talk*
☆ StkTokens: Enforcing Well-Bracketed Control Flow and Stack Encapsulation Using Linear Capabilities
Lau Skorstengaard , Dominique Devriese Vrije Universiteit Brussel, Belgium, Lars Birkedal Aarhus University

15:43 - 16:05 *Talk*
☆ Two sides of the same coin: Session Types and Game Semantics
Simon Castelan Imperial College London, UK, Nobuko Yoshida Imperial College London
DOI Pre-print

16:05 - 16:27 *Talk*
☆ Exceptional Asynchronous Session Types: Session Types without Tiers
Simon Fowler The University of Edinburgh, Sam Lindley University of Edinburgh, UK, J. Garrett Morris University of Kansas, USA, Sara Décova
Pre-print

16:37 - 17:43: Research Papers - Session Types II at POPL Track 2

16:37 - 16:59 *Talk*
☆ Interconnectability of Session-Based Logical Processes
TOPLAS
Bernardo Toninho NOVA-LINCS, FCT/UNL, Nobuko Yoshida Imperial College London
DOI Pre-print

16:59 - 17:21 *Talk*
☆ Distributed Programming using Role-Parametric Session Types in Go
David Castro Imperial College London, Raymond Hu Imperial College London, Sung-Shik Jongmans Open University of the Netherlands, Nicholas Ng Imperial College London, Nobuko Yoshida Imperial College London
DOI Pre-print

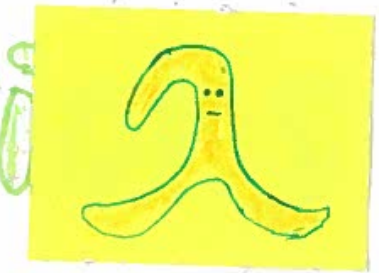
17:21 - 17:43 *Talk*
☆ Less is More: Multiparty Session Types Revisited
Alceste Scalas Imperial College London, Nobuko Yoshida Imperial College London
DOI Pre-print

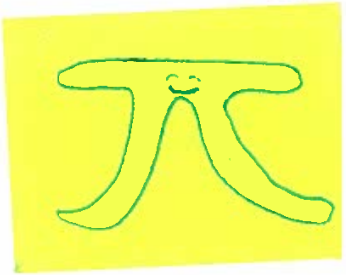
on Polymorphic Sessions and Functions or a Tale of Two Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

On Polymorphic Sessions and Functions



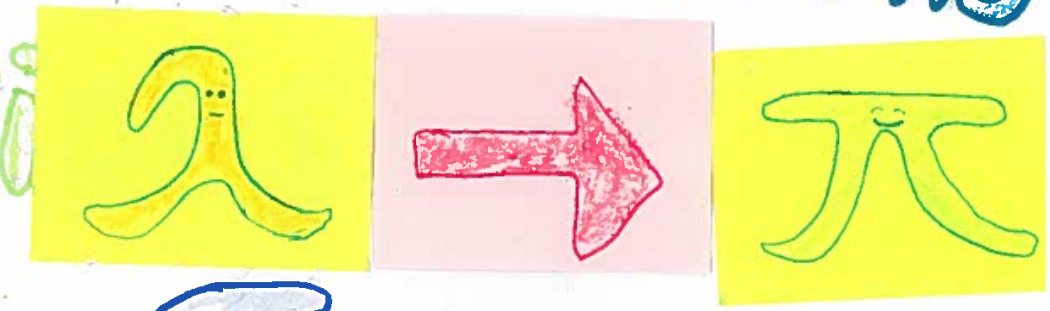


A Tale of Two Encodings

Fully Abstract

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 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

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

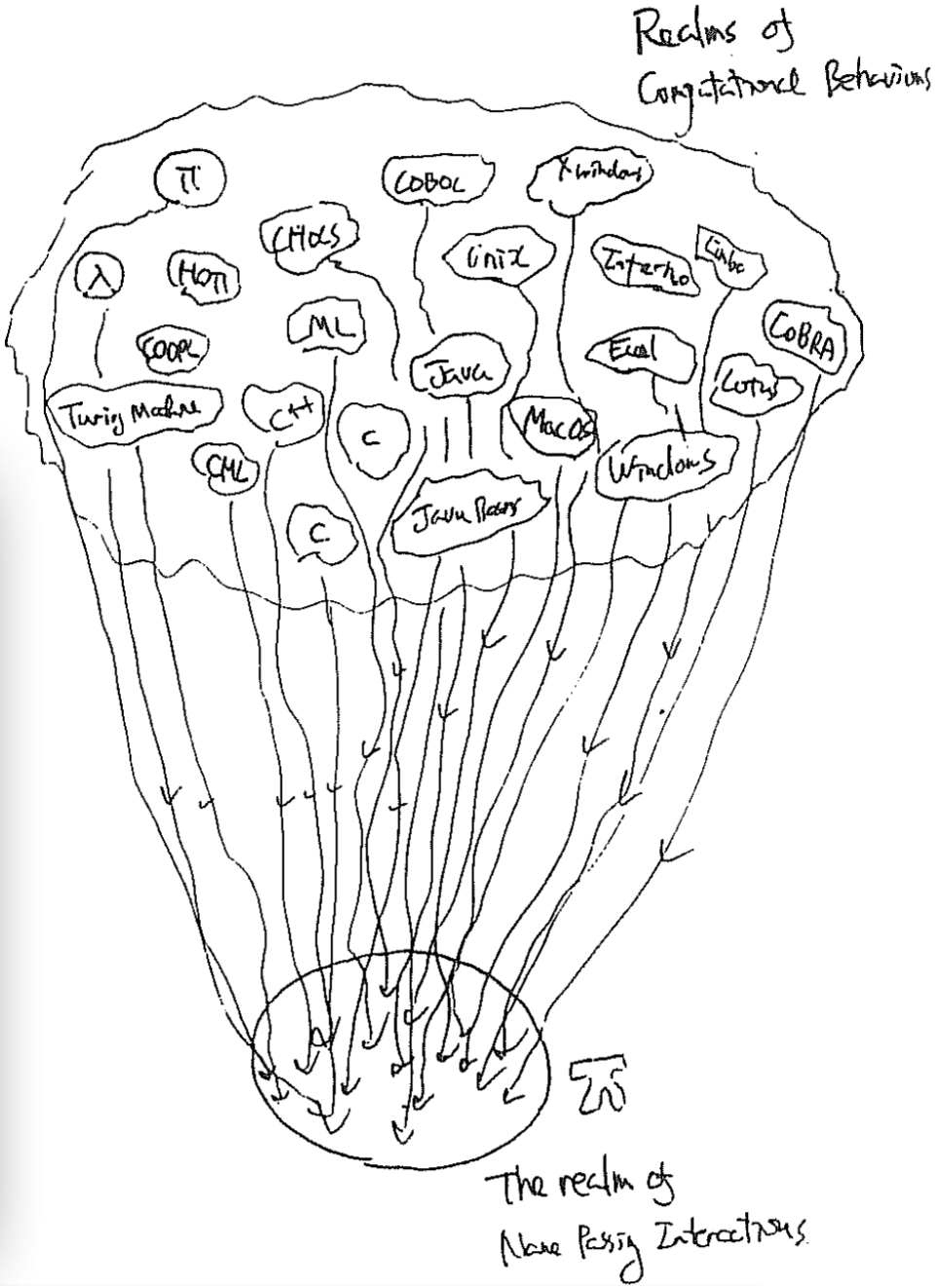
$[\![\lambda x.M]\!]_u \stackrel{\text{def}}{=} u(xu').[\![M]\!]_{u'}.$

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

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

* Examples of Representable Computation.

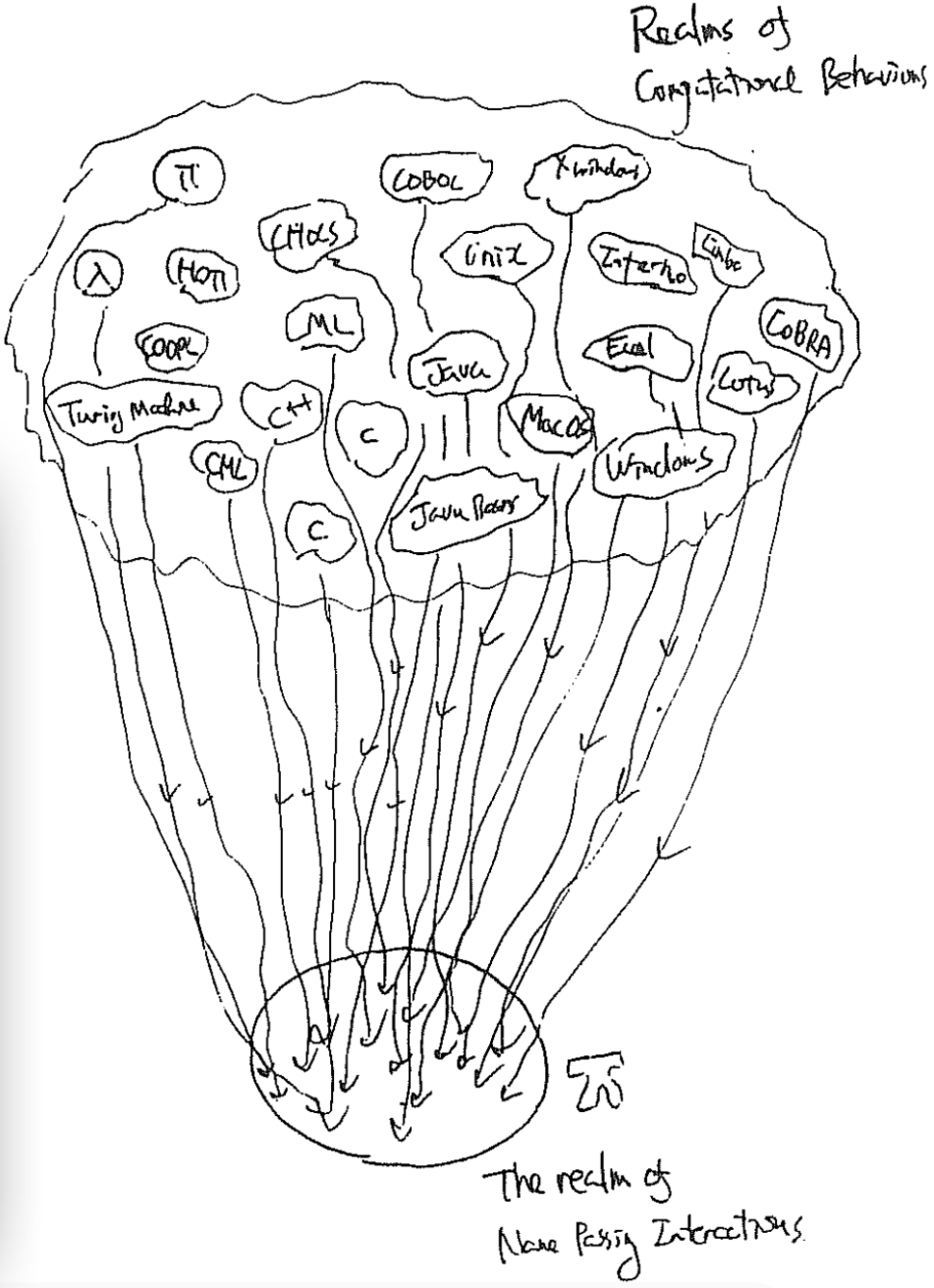
- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner92, ...]
- Proof Nets [Bellin and Scott93]
- Arbitrary "constant" interaction [HYS4]
- Strategies on Games [HO95]

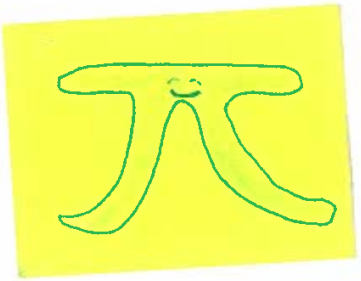


* Examples of Representable Computation.

Operationally Sound but NOT Fully Abstract

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner92, ...]
- Proof Nets [Bellin and Scott93]
- Arbitrary "constant" interaction [HYS4]
- Strategies on Games [HO95]



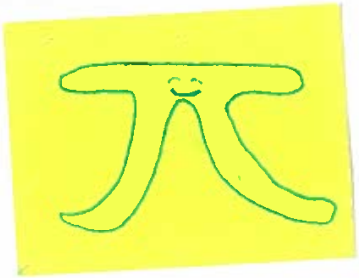


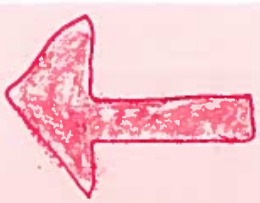
$M \simeq N$

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

Conextual Congruence

Contextual Congruence



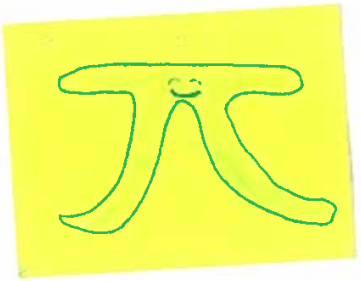
$M \simeq N$  $[\![M]\!] \simeq [\![N]\!]$

Conextual Congruence

Contextual Congruence

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

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

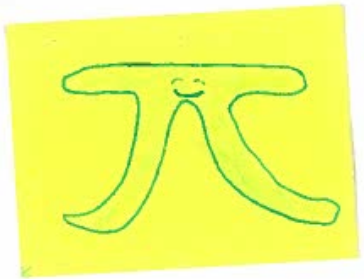


$M \simeq N$ 

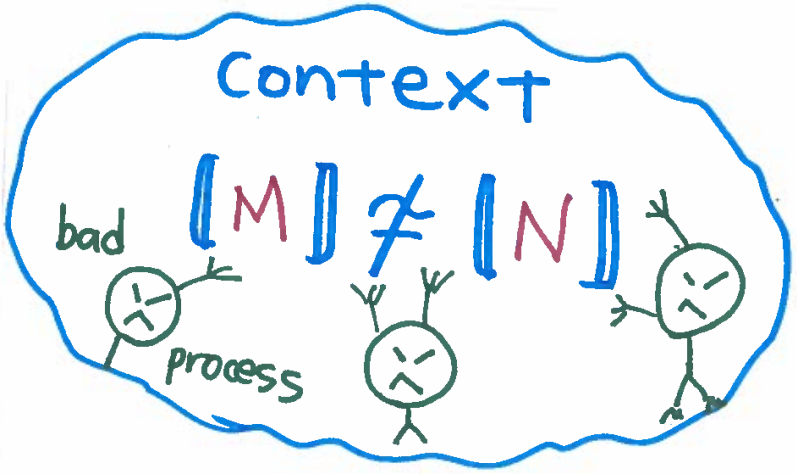
$[\![M]\!] \simeq [\![N]\!]$

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

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

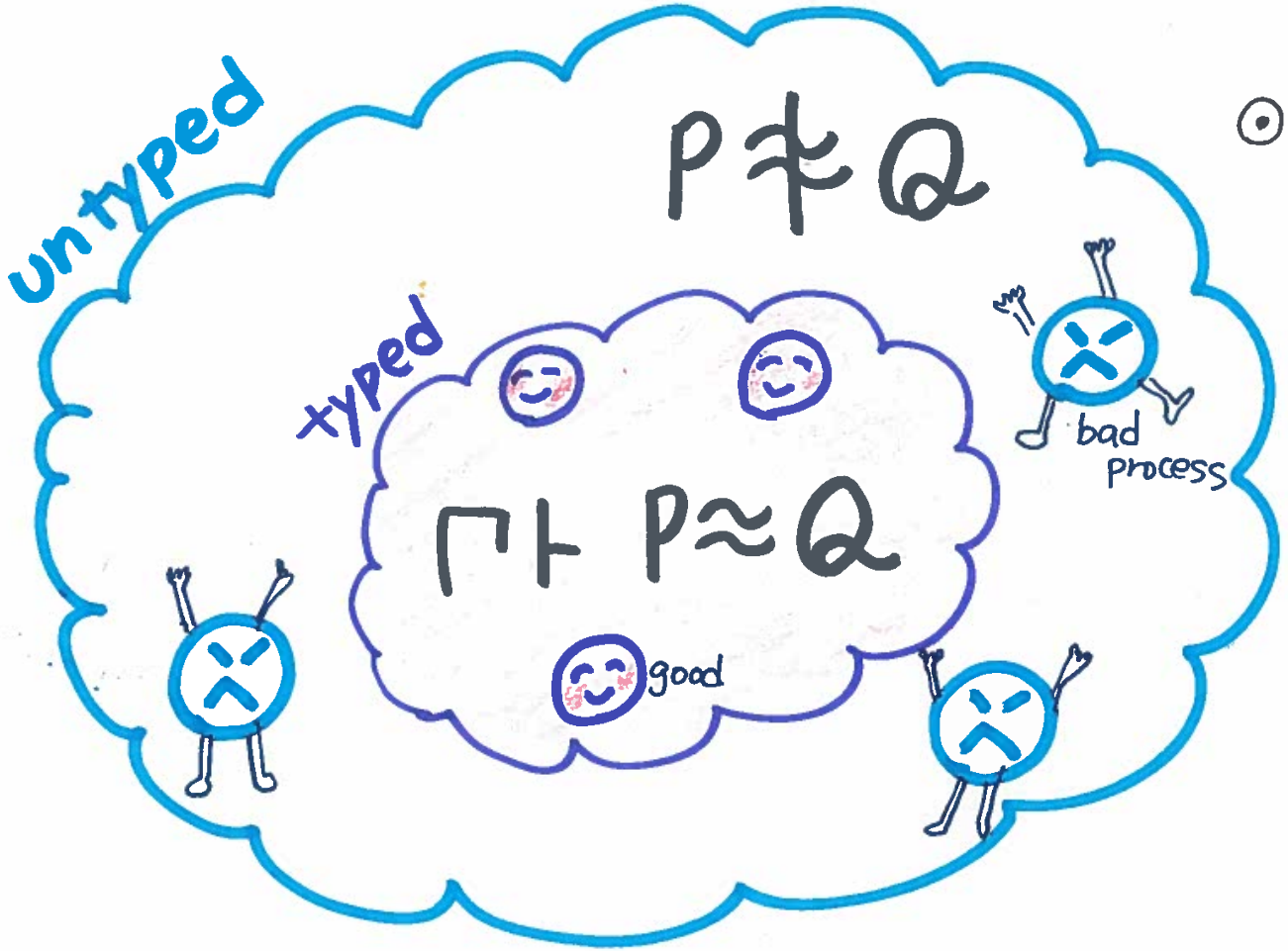


$M \simeq N$ NOT $[\![M]\!] \simeq [\![N]\!]$



Typed Semantics in π 1991 →

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

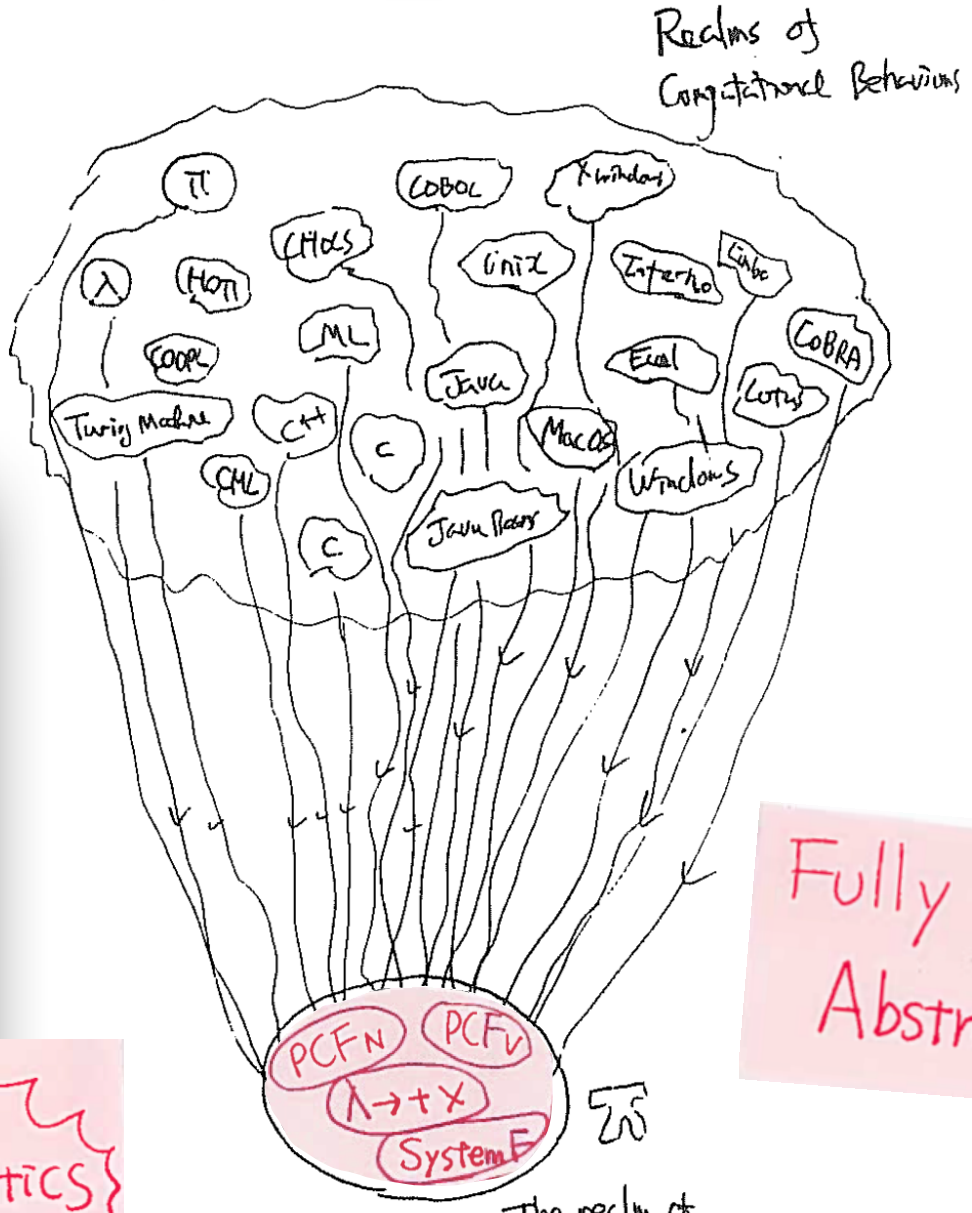


- ⊙ Correctness of Encoding
- ⊙ Limit environment Γ
  ⇒ Equate more processes
- ⊙ Compositional

* Examples of Representable Computation.

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner92, ...]
- Proof Nets [Bellin and Scott93]
- Arbitrary "constant" interaction [HYS4]
- Strategies on Games [HO95]
- ⋮

Game Semantics



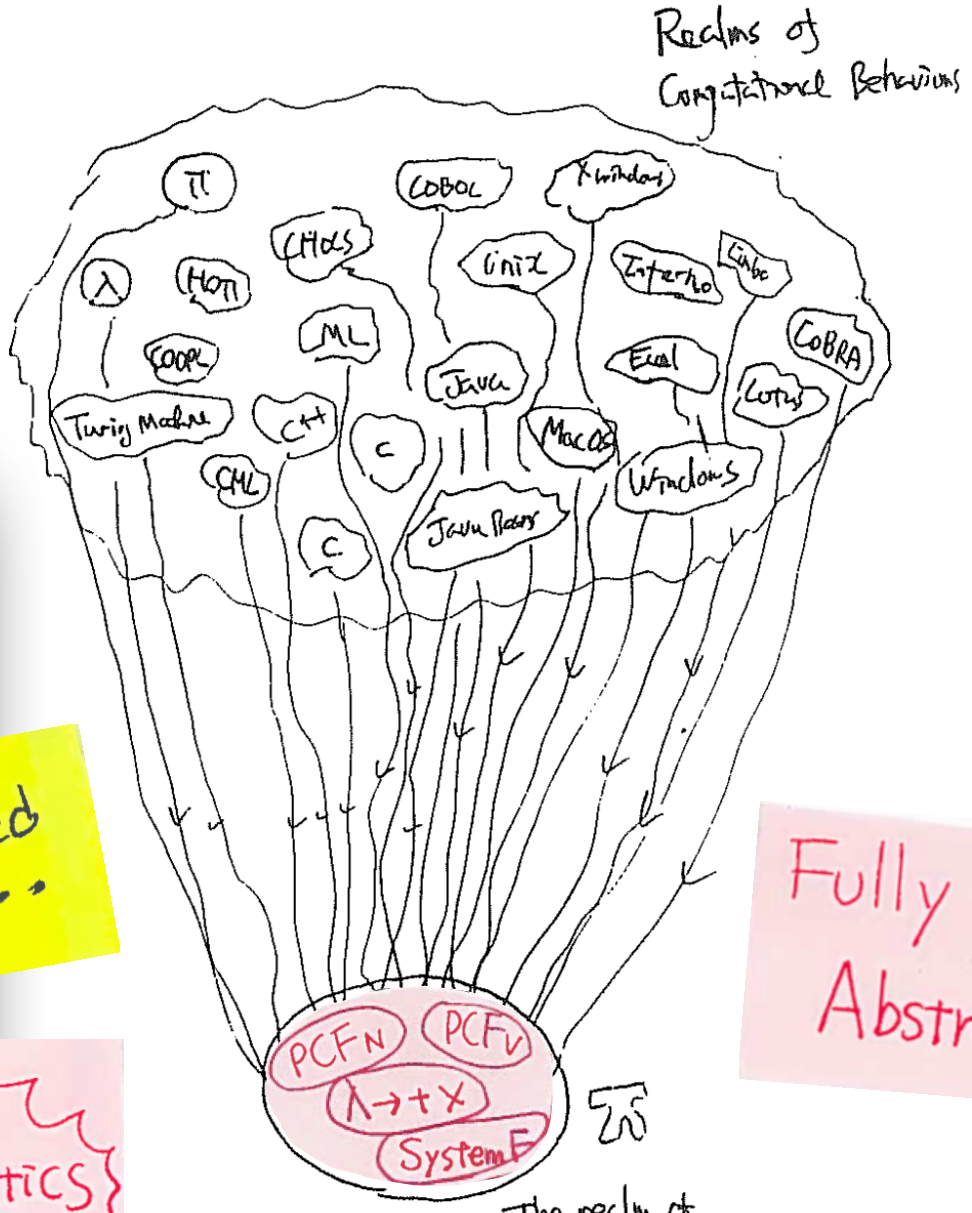
Fully Abstract

* Examples of Representable Computation.

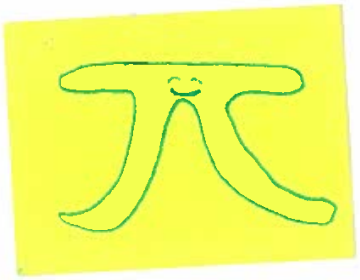
- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner
- Proof Nets [Bellin and Scott 93]
- Arbitrary "constant" interaction [
- Strategies on Games [HO95]

Complicated ...

Game Semantics

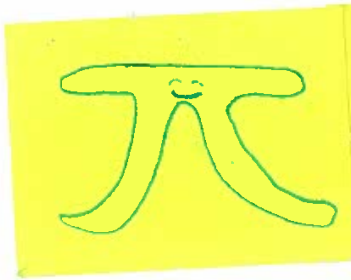


Fully Abstract



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

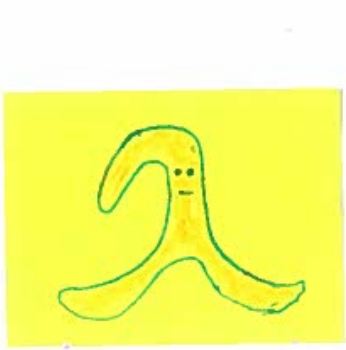
System F



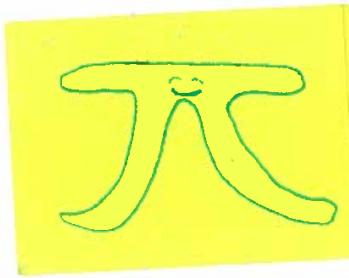
Session Π

$M \simeq N$

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



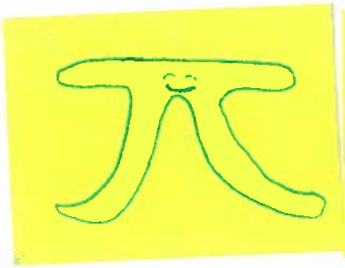
System F



Session Π

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

System F

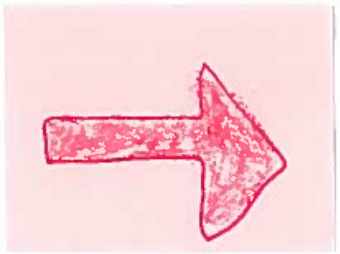


Session Π

$M \simeq N$

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

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

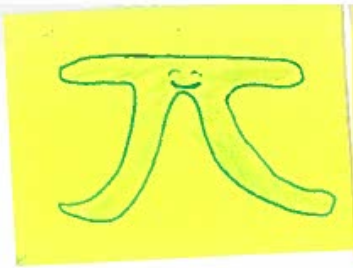


$P \simeq Q$

Reverse

New

System F



Session

$M \simeq N$

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

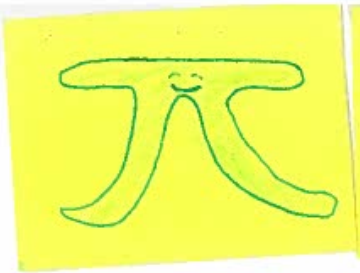
$[\![P]\!] \simeq [\![Q]\!]$

$P \simeq Q$

Reverse

New

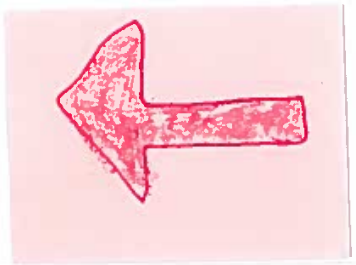
System F



Session Π

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

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



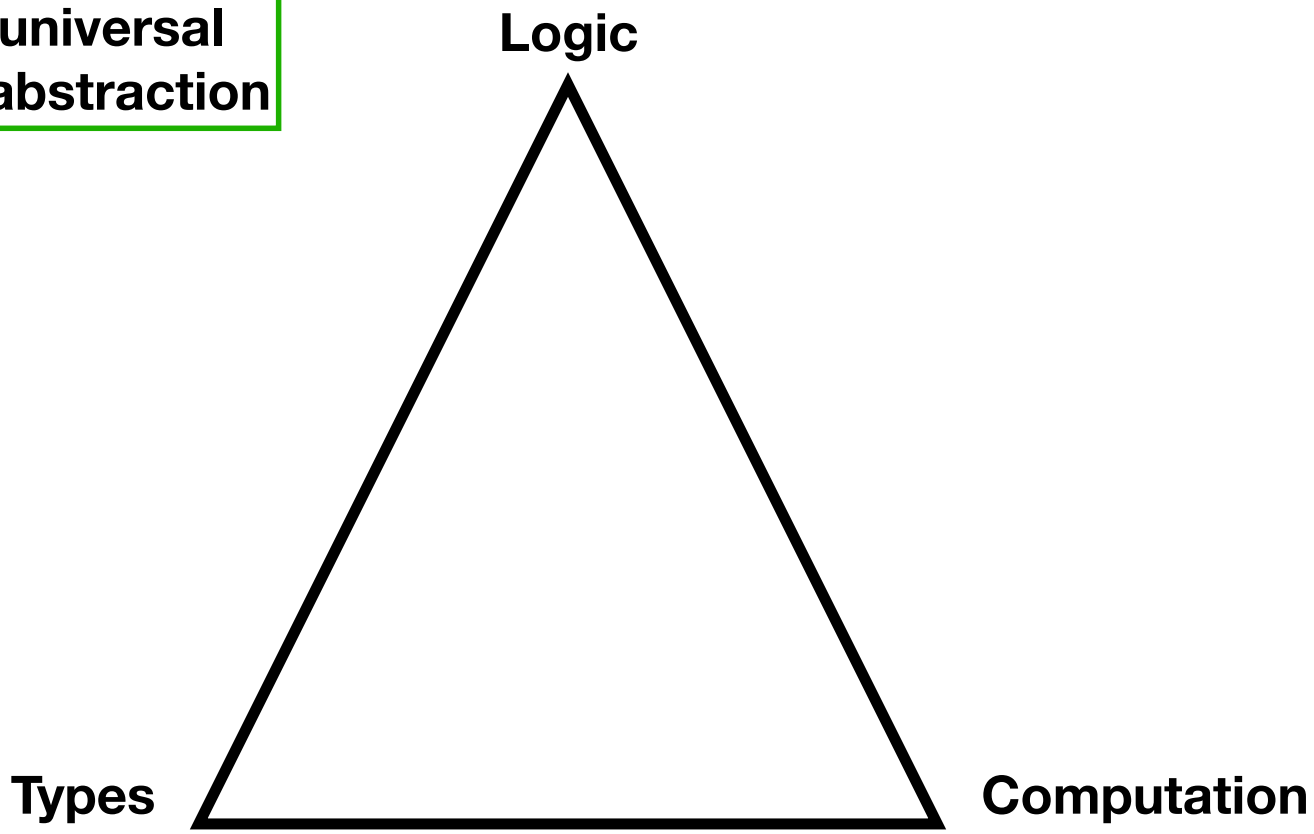
TWO APPLICATIONS

Frank Pfenning

A Rehabilitation of Message-Passing Concurrency

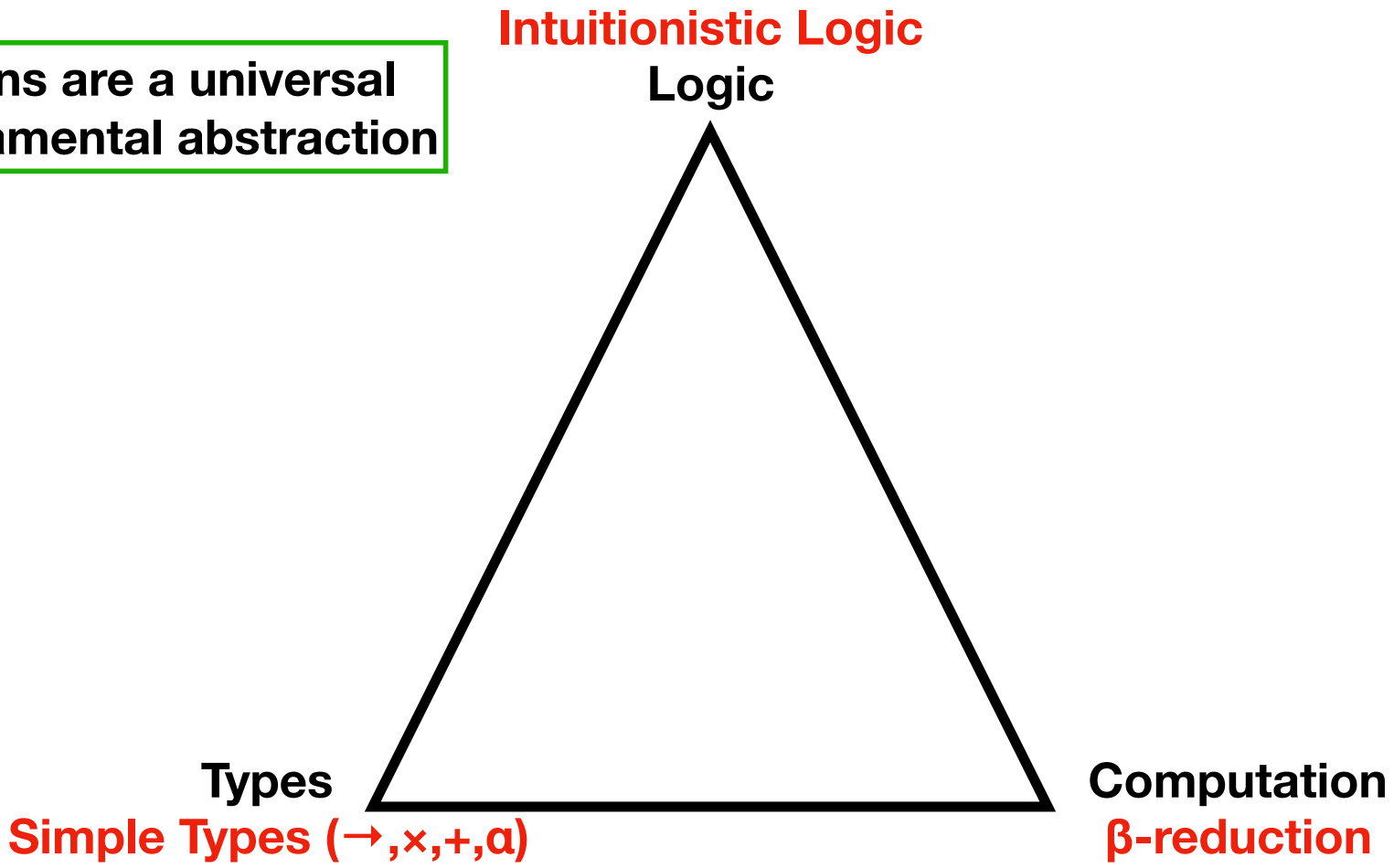
Why is Functional Programming So Effective?

Functions are a universal and fundamental abstraction



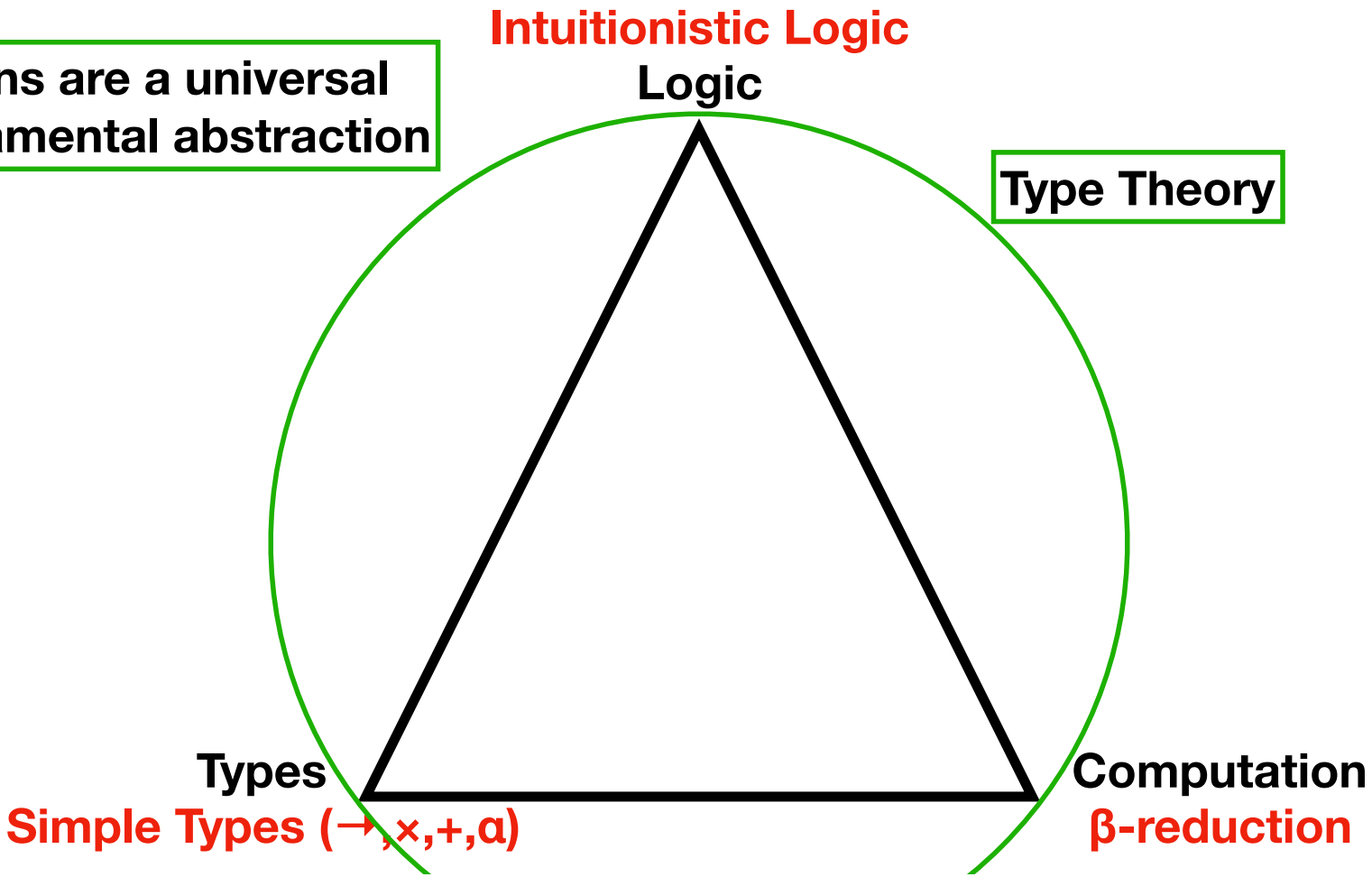
Why is Functional Programming So Effective?

Functions are a universal and fundamental abstraction



Why is Functional Programming So Effective?

Functions are a universal and fundamental abstraction



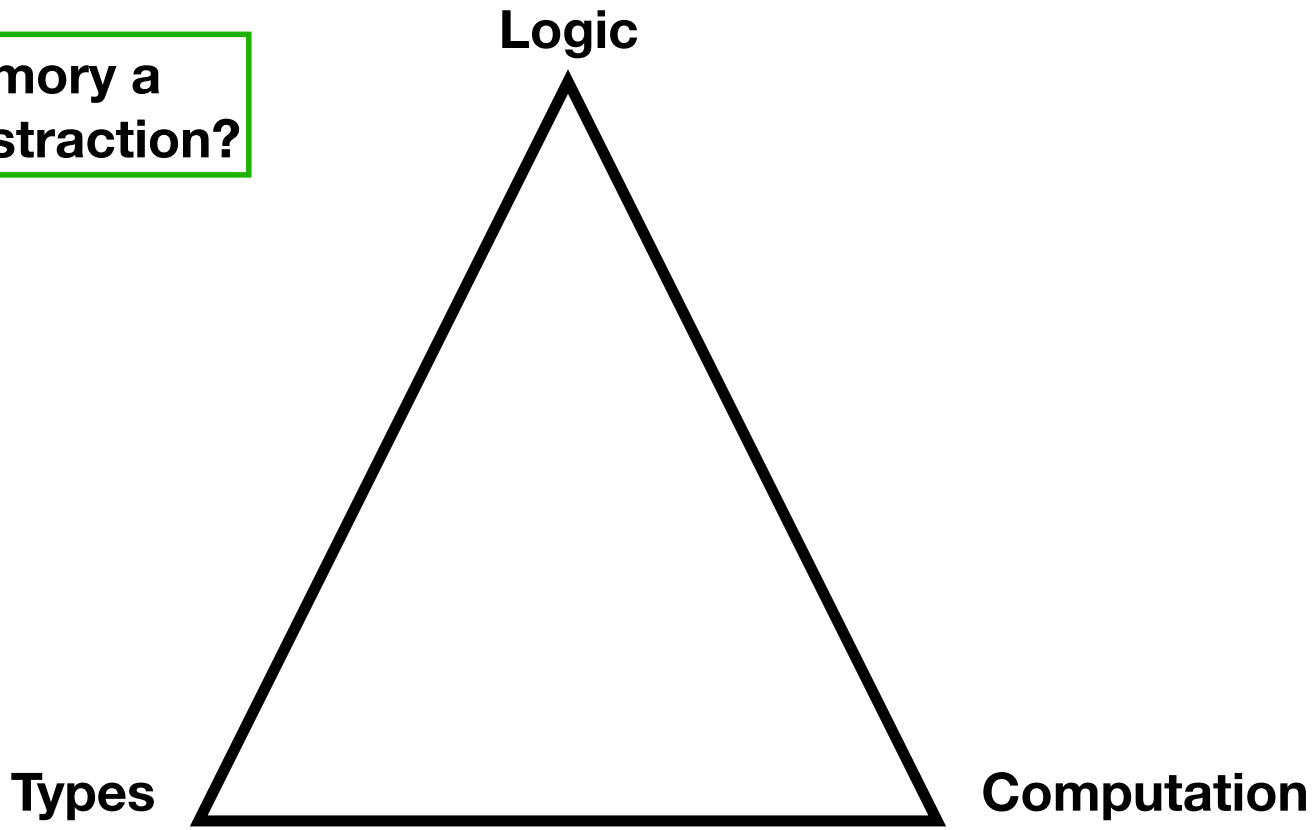
What about Concurrency?

What about Concurrency?

Is Shared Memory a Fundamental Abstraction?

What about Concurrency?

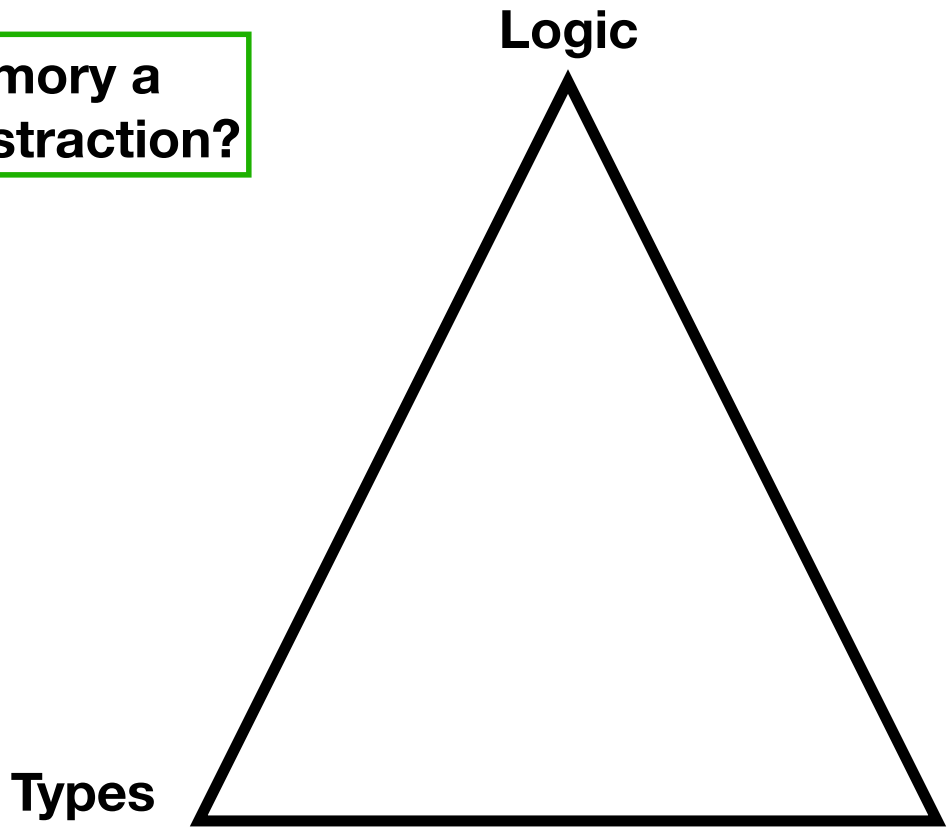
Is Shared Memory a Fundamental Abstraction?



What about Concurrency?

Concurrent Separation Logic

Is Shared Memory a Fundamental Abstraction?



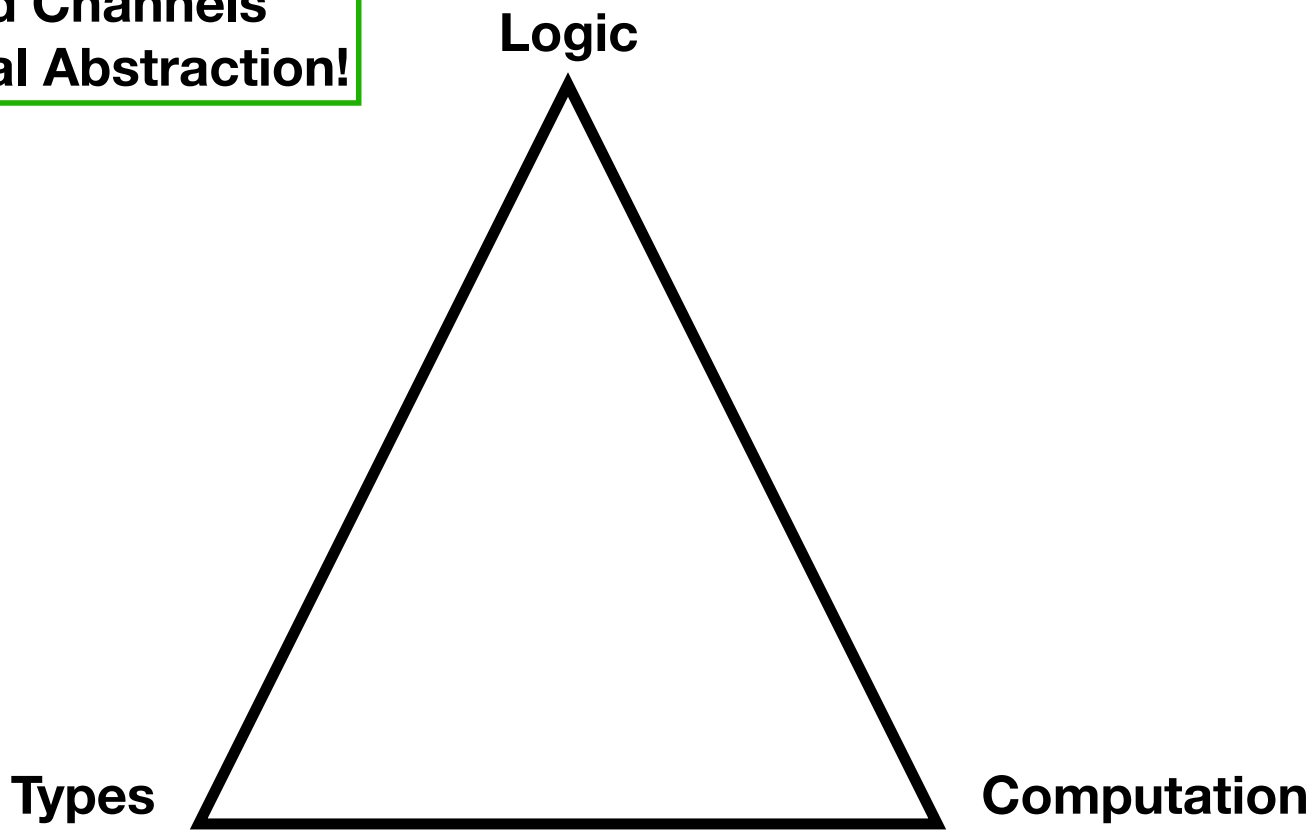
What about Concurrency?

What about Concurrency?

Processes and Channels
are a Fundamental Abstraction!

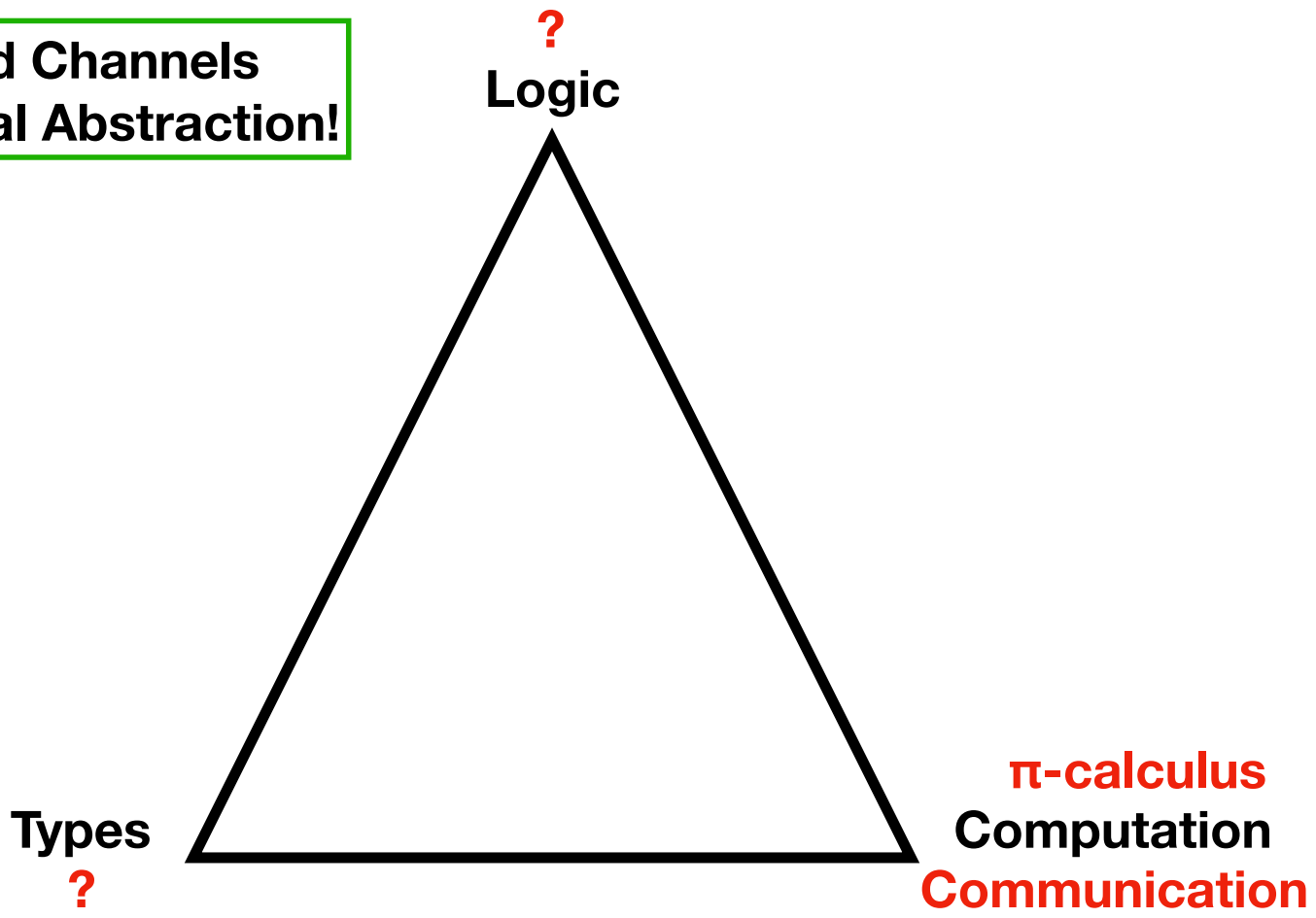
What about Concurrency?

Processes and Channels are a Fundamental Abstraction!



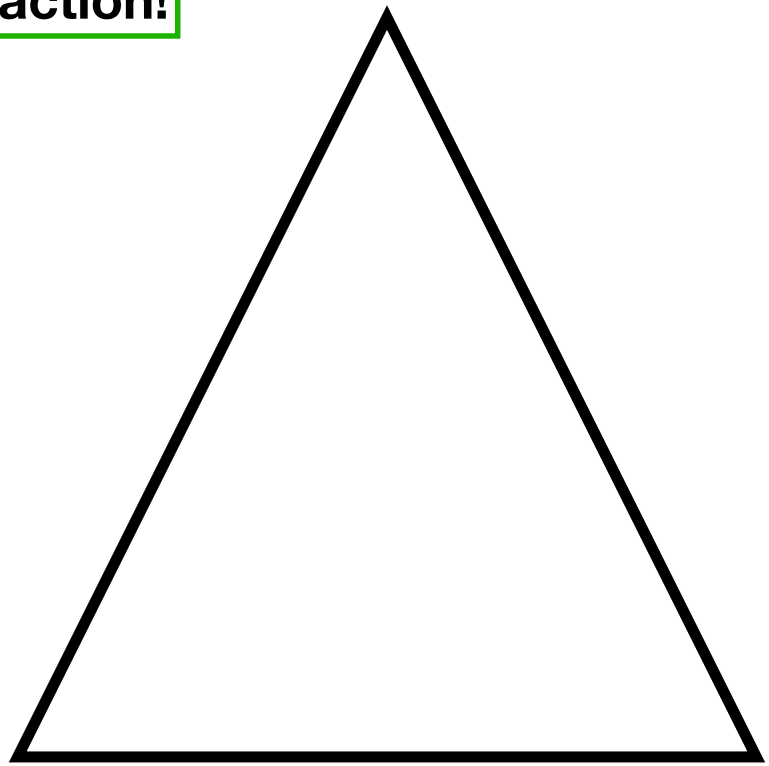
What about Concurrency?

Processes and Channels are a Fundamental Abstraction!



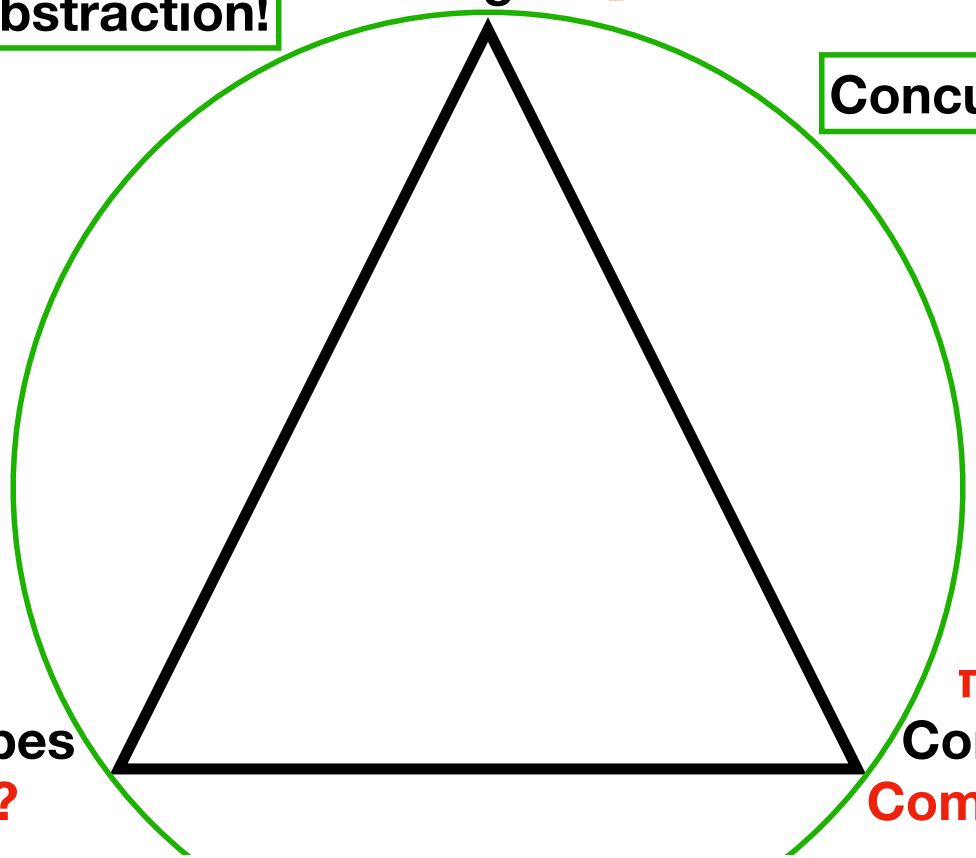
What about Concurrency?

Processes and Channels are a Fundamental Abstraction!



What about Concurrency?

Processes and Channels
are a Fundamental Abstraction!



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

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

Types as Propositions

Types as Propositions

- These types are exactly the propositions of linear logic, except ‘!’

Types as Propositions

- These types are exactly the propositions of linear logic, except ‘!’
- An instance of the Curry-Howard correspondence

Types as Propositions

- These types are exactly the propositions of linear logic, except ‘!’
- An instance of the Curry-Howard correspondence
  - Typing rules correspond to sequent calculus inference rules

Types as Propositions

- These types are exactly the propositions of linear logic, except ‘!’
- An instance of the Curry-Howard correspondence
  - Typing rules correspond to sequent calculus inference rules
  - Programs correspond to process expressions

Types as Propositions

- These types are exactly the propositions of linear logic, except ‘!’
- An instance of the Curry-Howard correspondence
  - Typing rules correspond to sequent calculus inference rules
  - Programs correspond to process expressions
  - Communication corresponds to cut reduction

Judgement

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

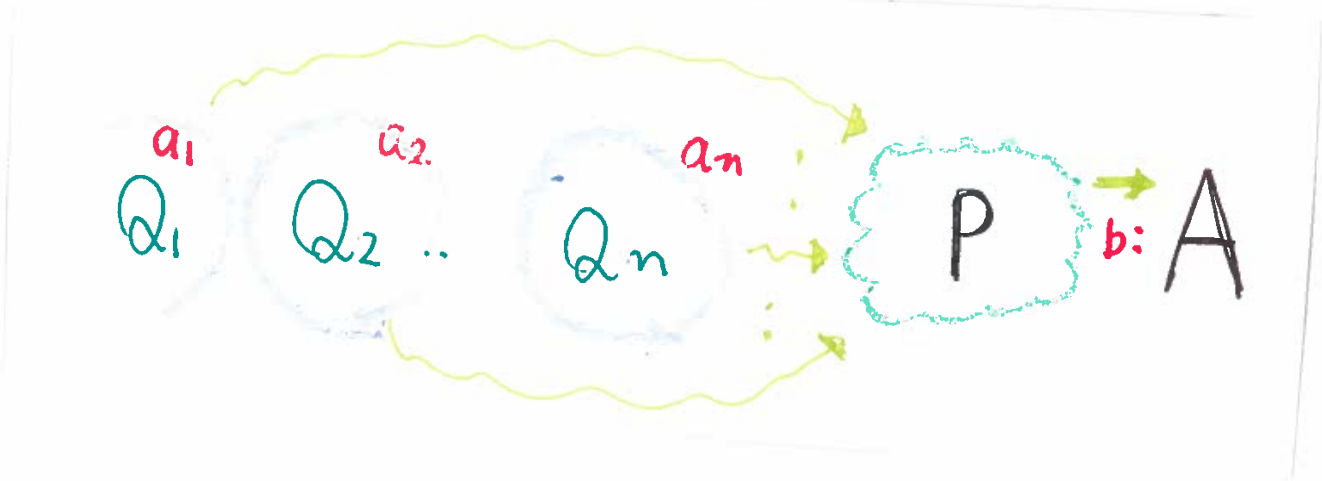
Process

Poly Vars

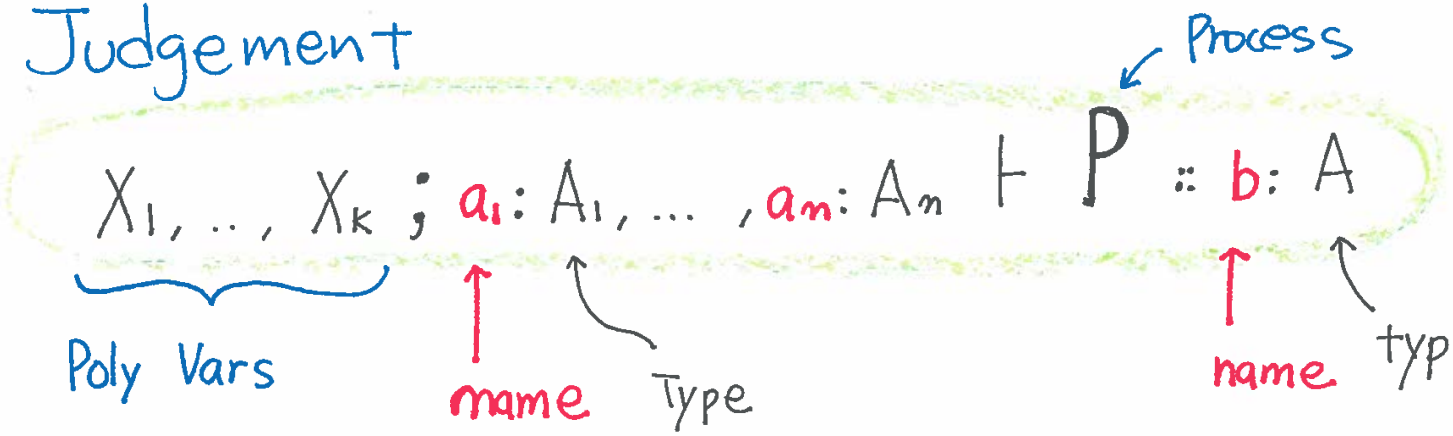
name — Type

name — type

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

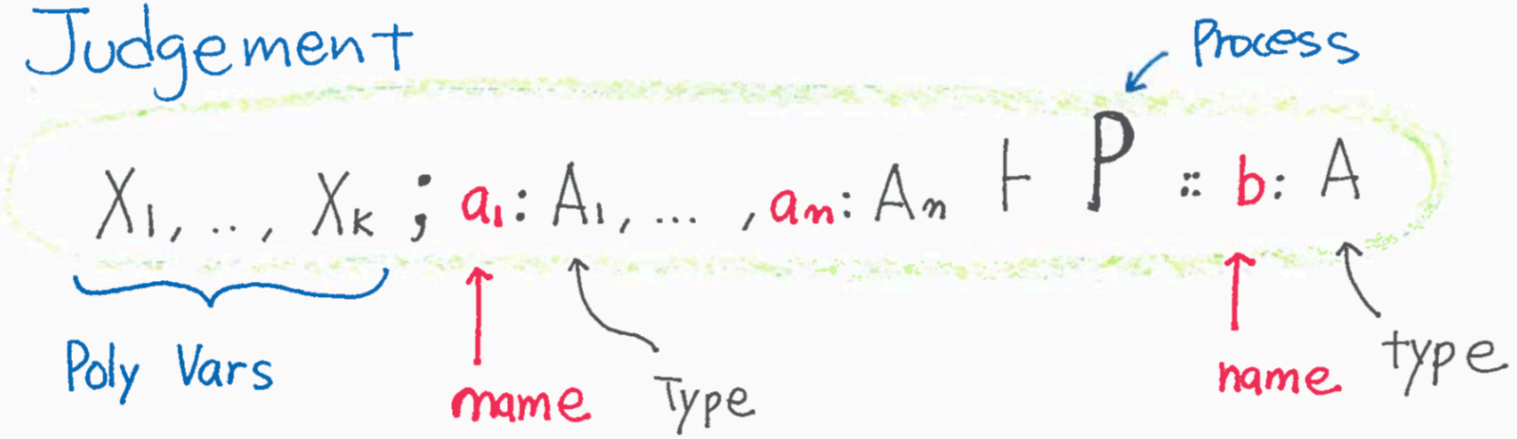


Judgement



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

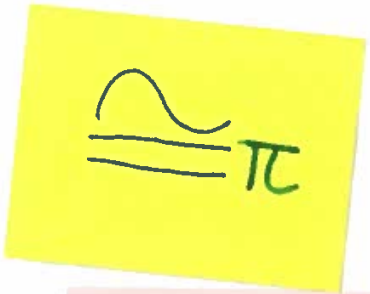
Polymorphic Session

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 λ to π

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 ⇒ Right / Elim ⇒ 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

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

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

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

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 ⇒ Right / Elim ⇒ Left + Cut + Identity

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

↑ name

λ in π

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

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

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

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

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

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

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

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

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

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

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

$$(\!| [\![M]\!]_Z |\!) \cong M \qquad \forall P\ \exists M\ [\![M]\!]_Z \cong P$$

$$[\![(\!| P |\!)]\!]_Z \cong P \qquad \forall M\ \exists P\ (\!| P |\!) \cong M$$

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

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

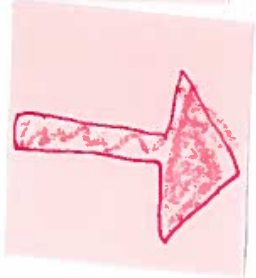
$$[\![(\!|\, P \,|\!)]\!]_z \cong P$$

Full Abstraction

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

$$M \cong N$$

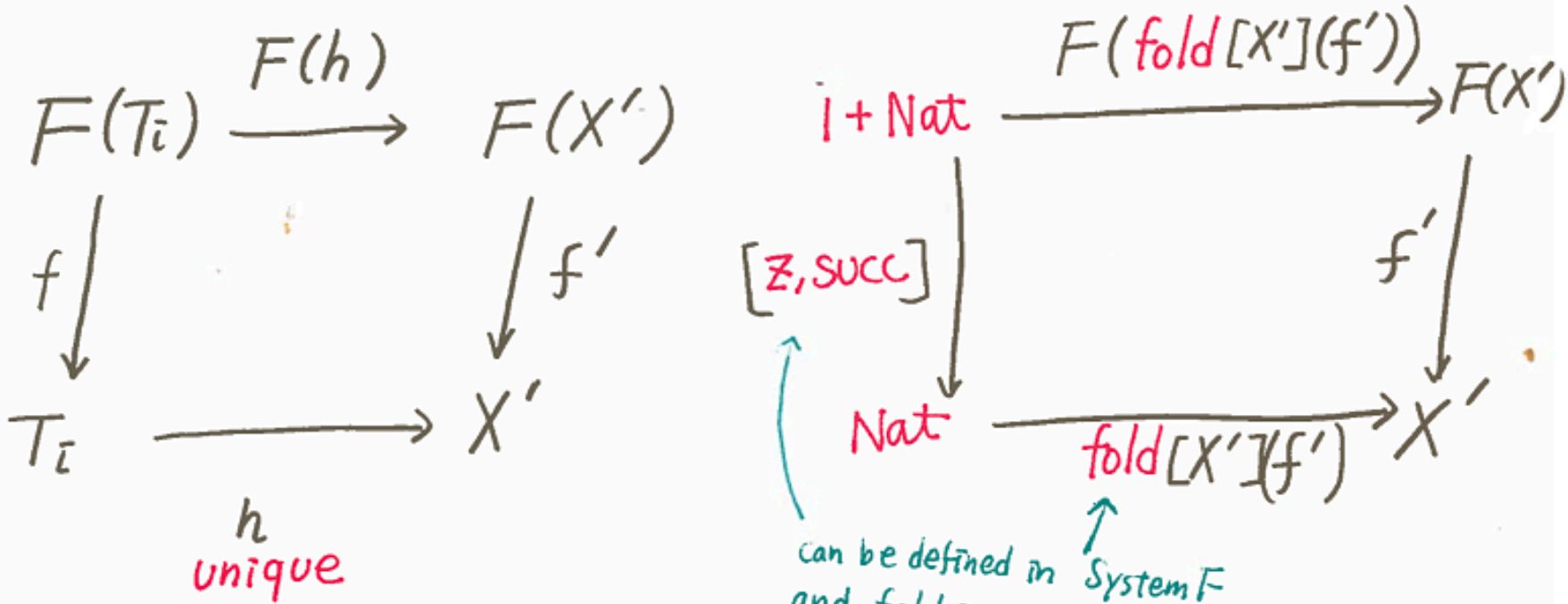
$$(\!|\, P \,|\!) \cong (\!|\, Q \,|\!)$$



$$P \cong Q$$

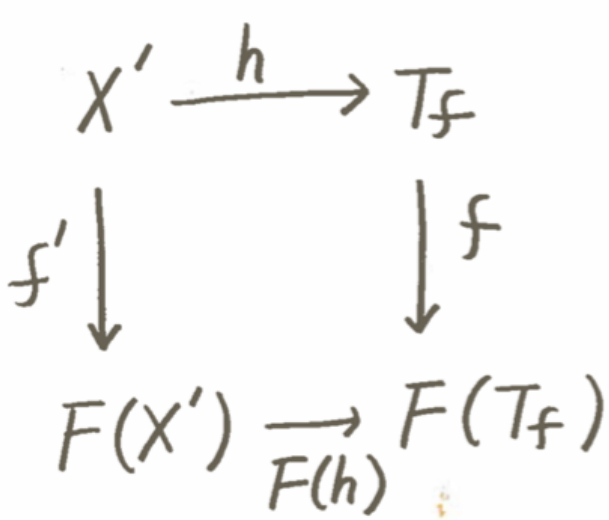
Application 1 Inductive and Coinductive Types

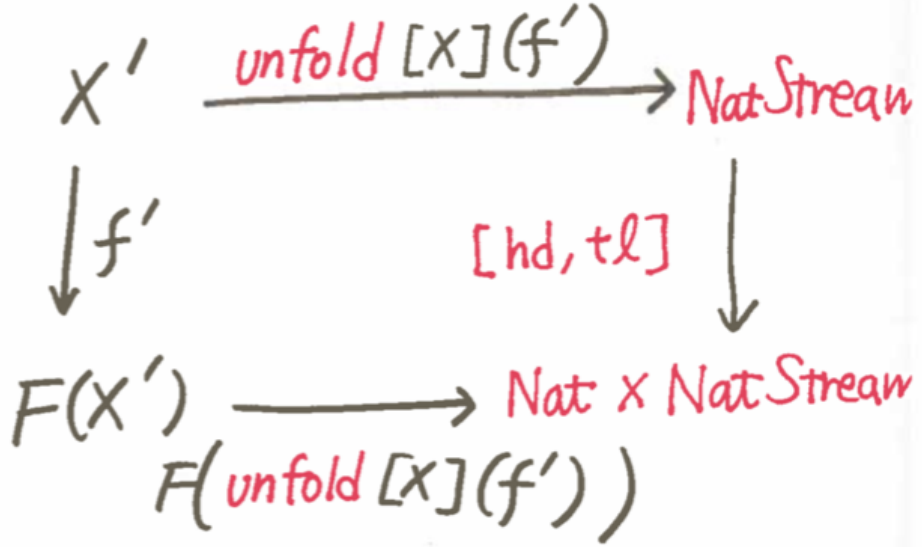
Parametric Poly is Expressive Enough to Encode Inductive/Co-Inductive Types as Initial / Final (Co)Algebra



Inductive

Coinductive Types





Question Sess Poly π is Expressive Enough?

Theorems

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

$\forall_A\, Q$ 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

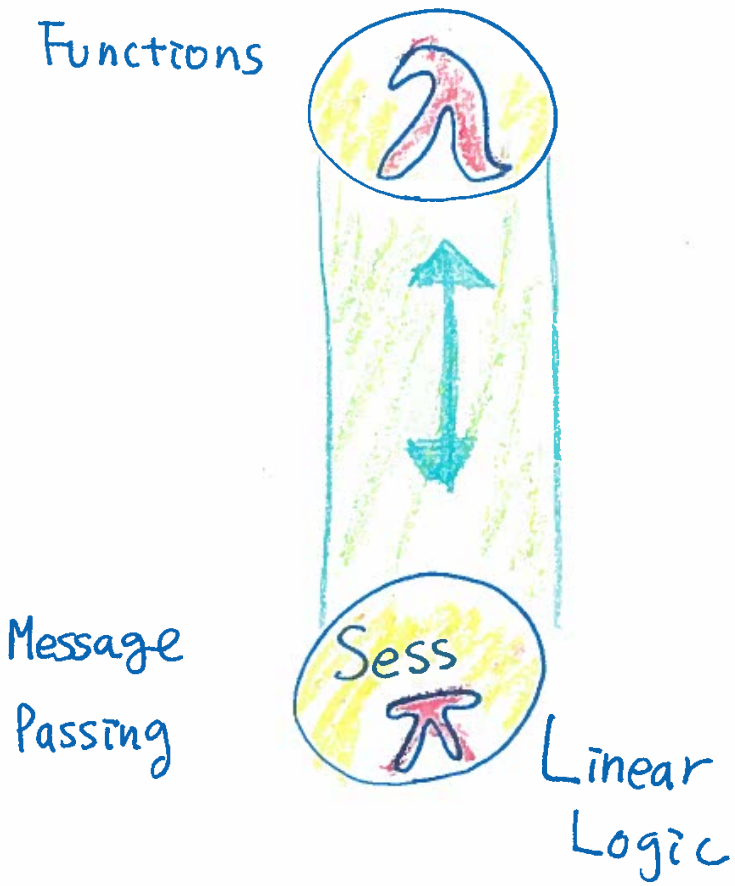
- Full Abstraction / Inverse $x\langle M\rangle.P$
- Strong Normalisation via Strong Normalisation

HOπ : λ

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

cf. [CPPT'13] logical relation

Summary

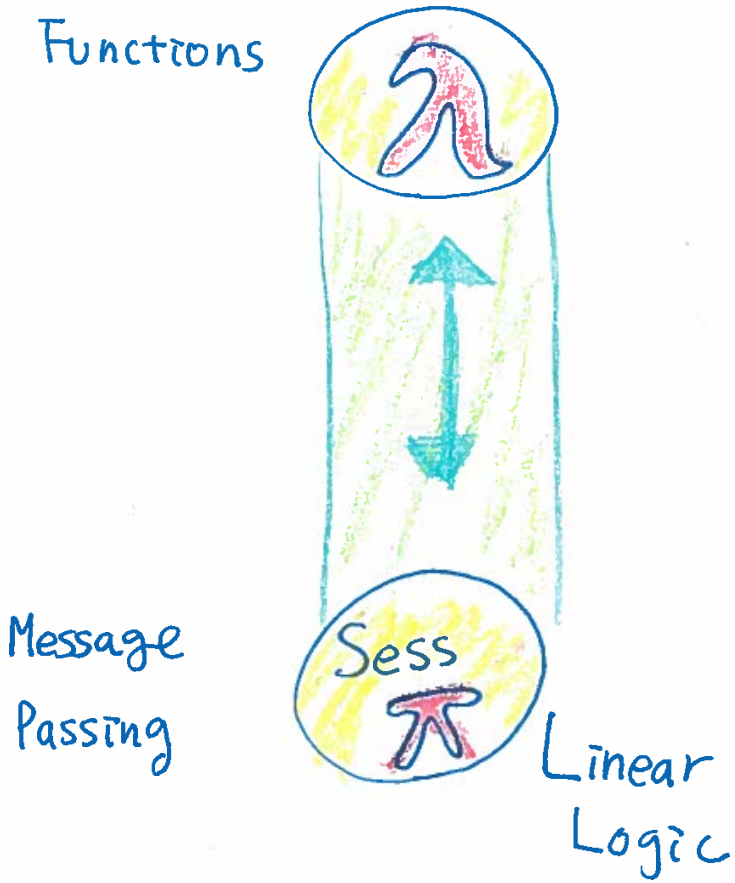


LL-based

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

Summary

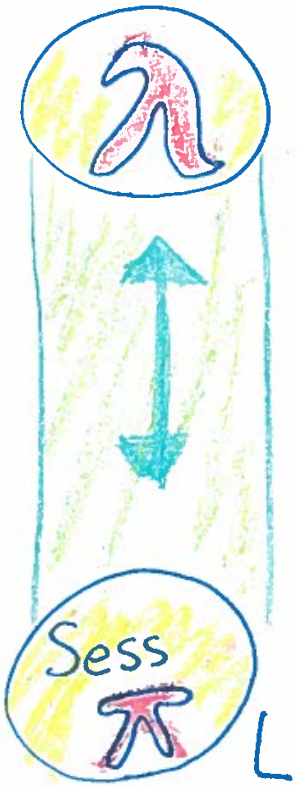
SAD?



LL-based

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

Summary



LL-based

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