on Polymorphic Sessions and Functions or a Tale of Two Encodings

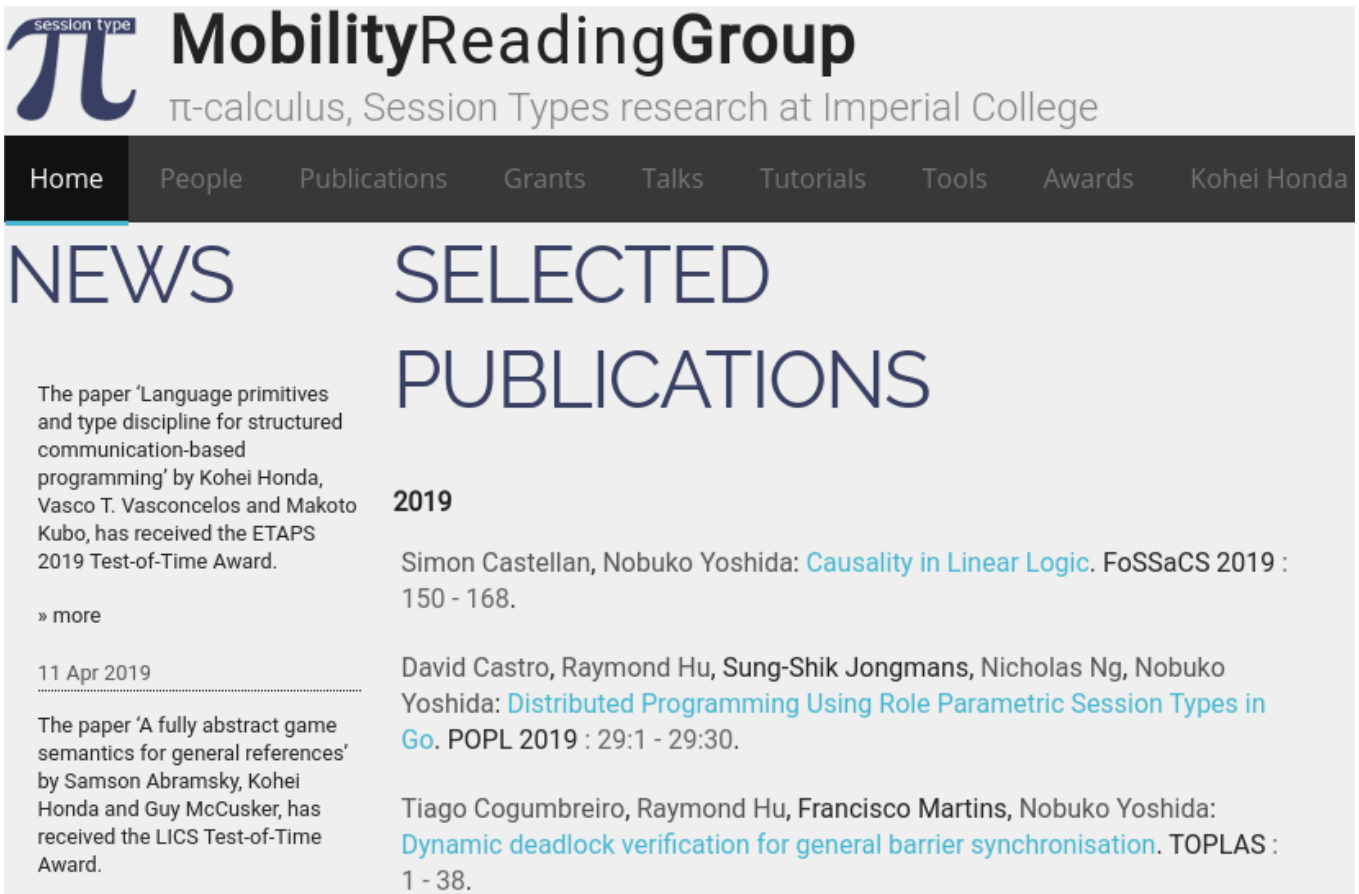
Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

The Mobility Reading Group at Imperial College

Our research group is **specialised in π-calculus** and **session types** — both **theory** and **applications**

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



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

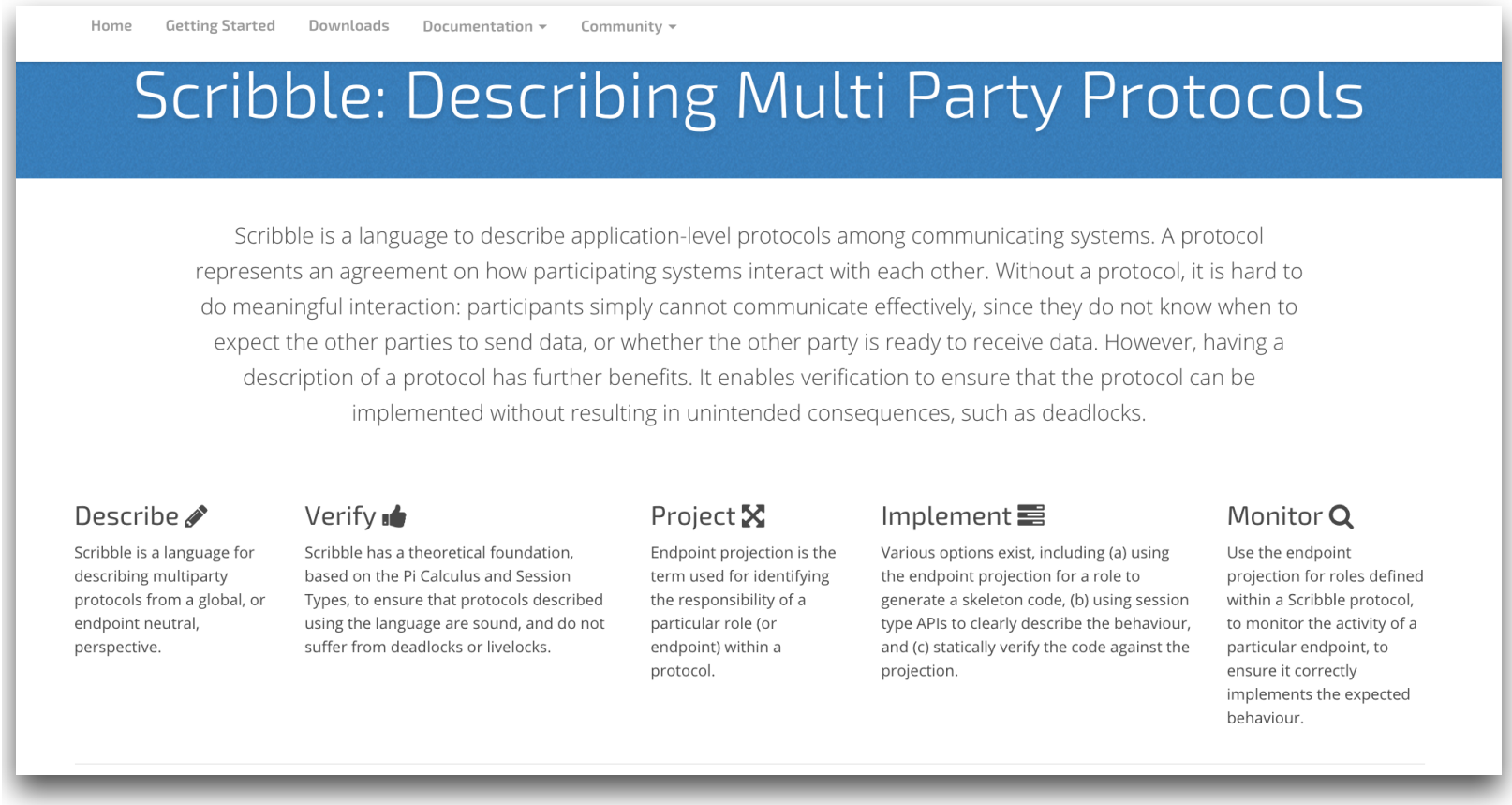
Guy McCusker
St John's College, Oxford
mccusker@comlab.ox.ac.uk

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.

Impacts (I): Scribble

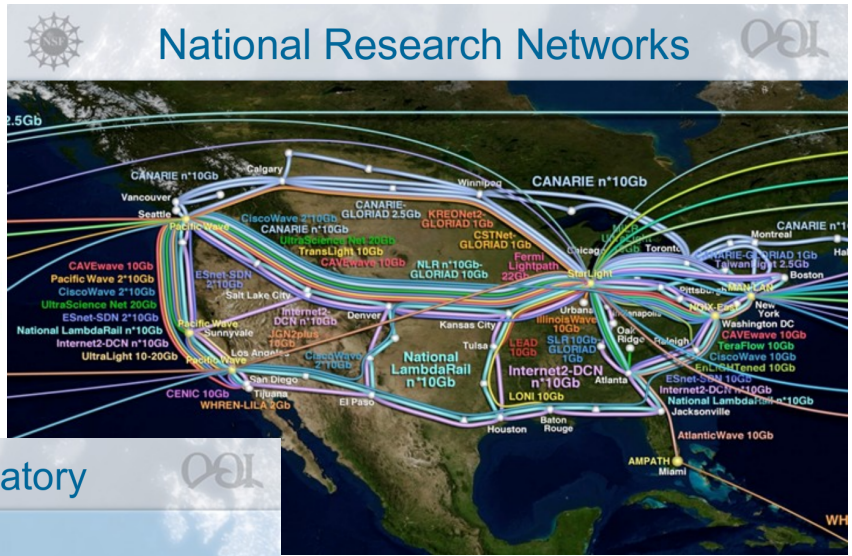
www.scribble.org

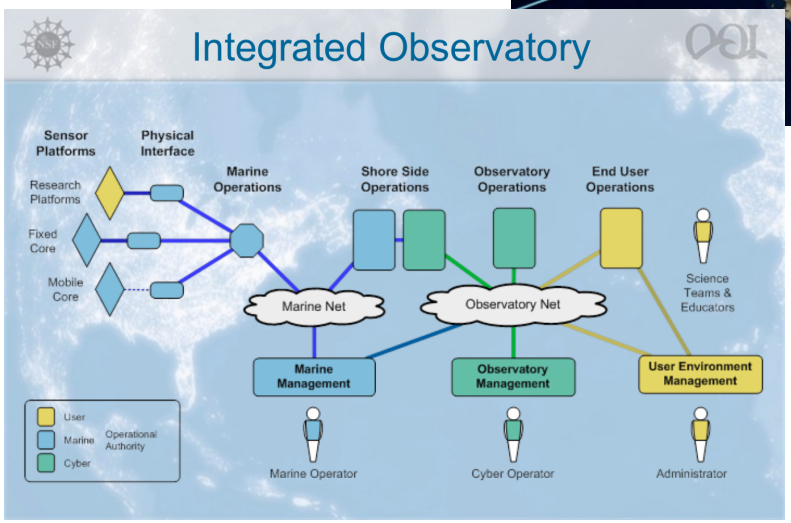


Impacts (II): Ocean Observatories Initiative

Collaboration: use session types on top of OOI network to **guarantee global safety**

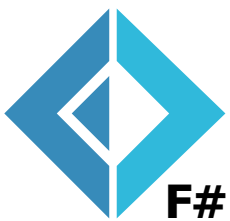
OCEAN OBSERVATORIES INITIATIVE



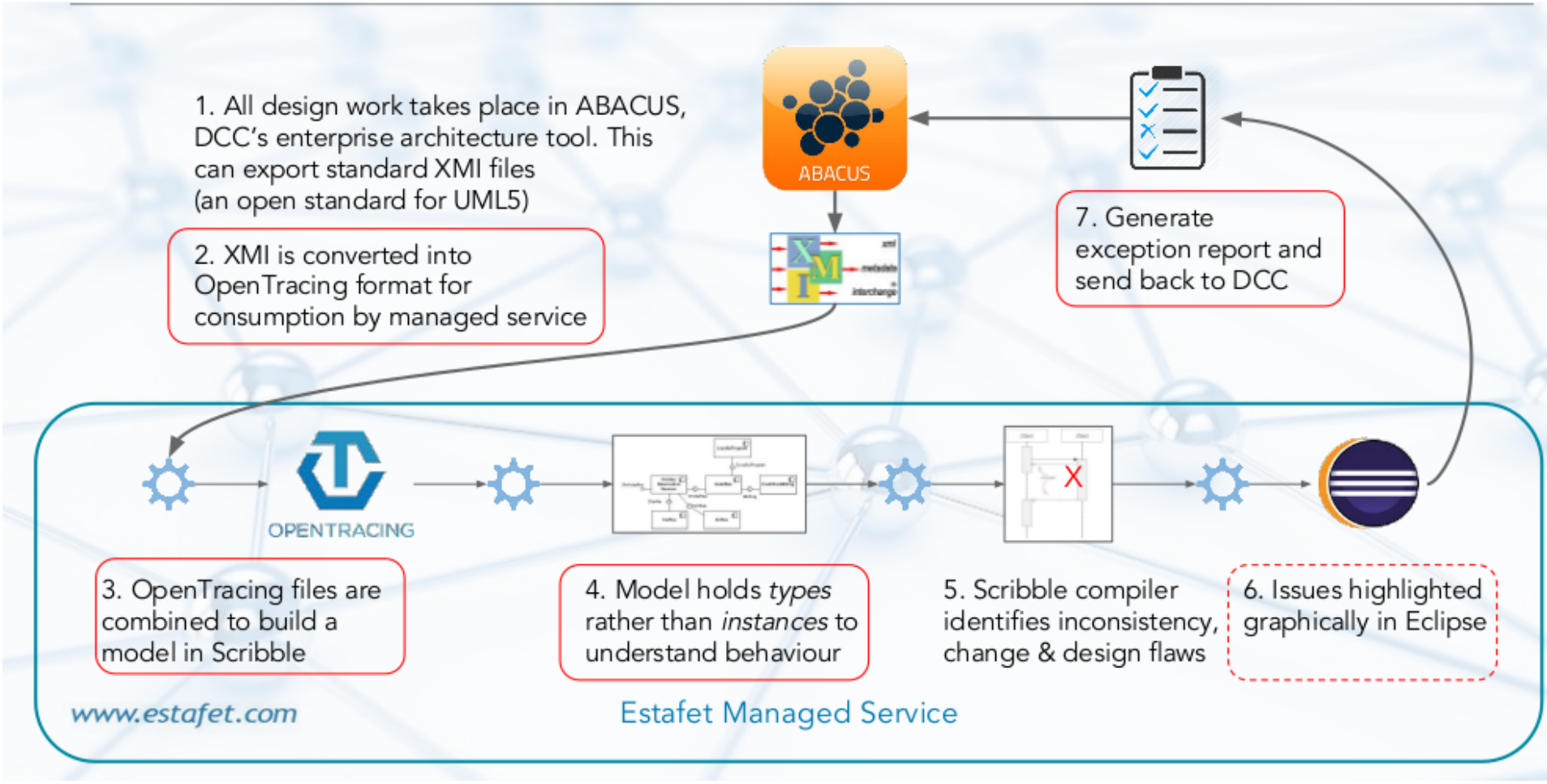


Session types in programming languages

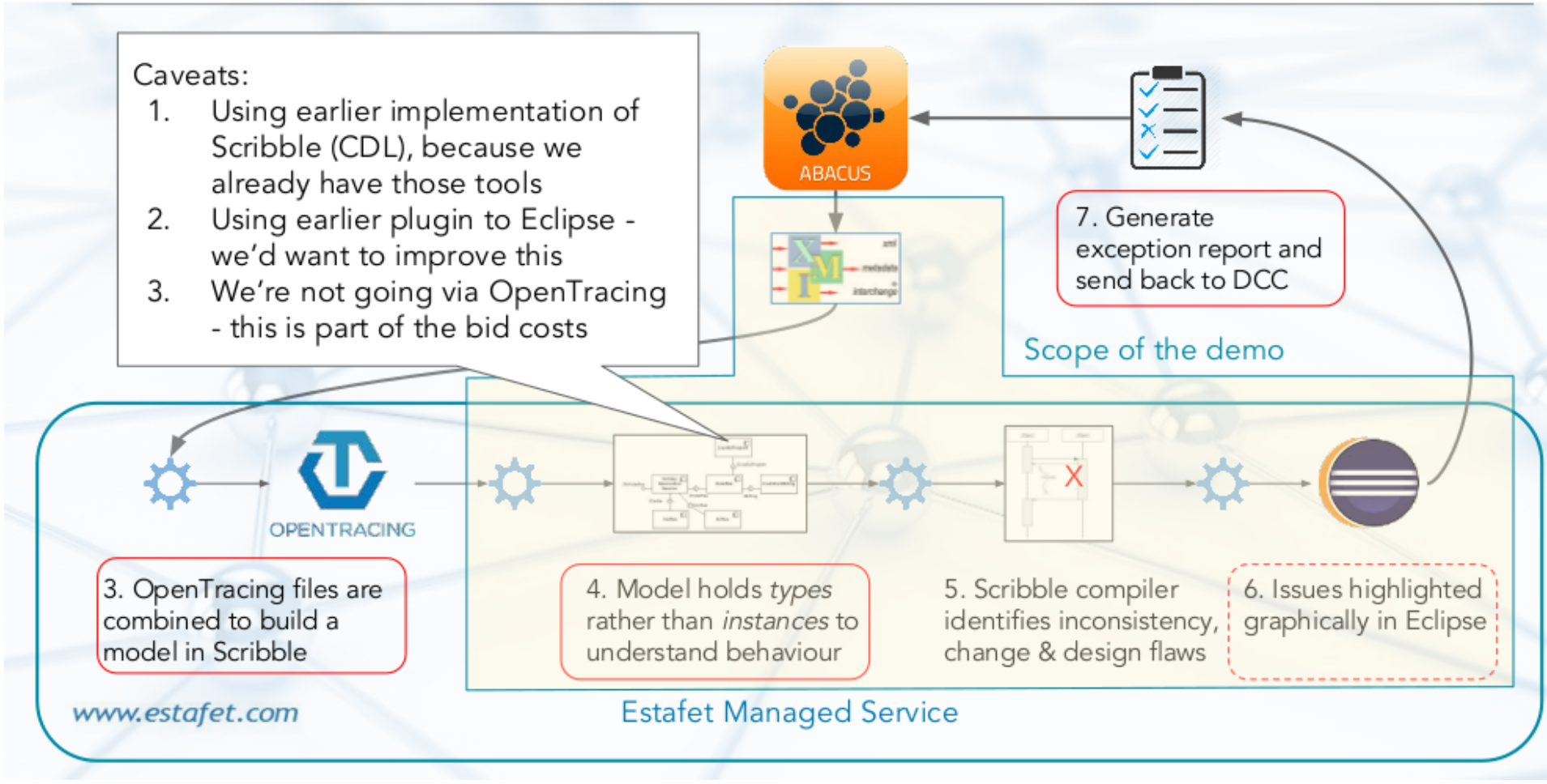
(Elements of) the **multiparty session types** theory have been **implemented in many languages and libraries**, e.g.:
Java, Go, Python, Scala, C, MPI-C, F#, F⋆, Erlang, Haskell, OCaml, Dotty3, PureScript, TypeScript and Rust.



End-to-End Switching Programme by DCC



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

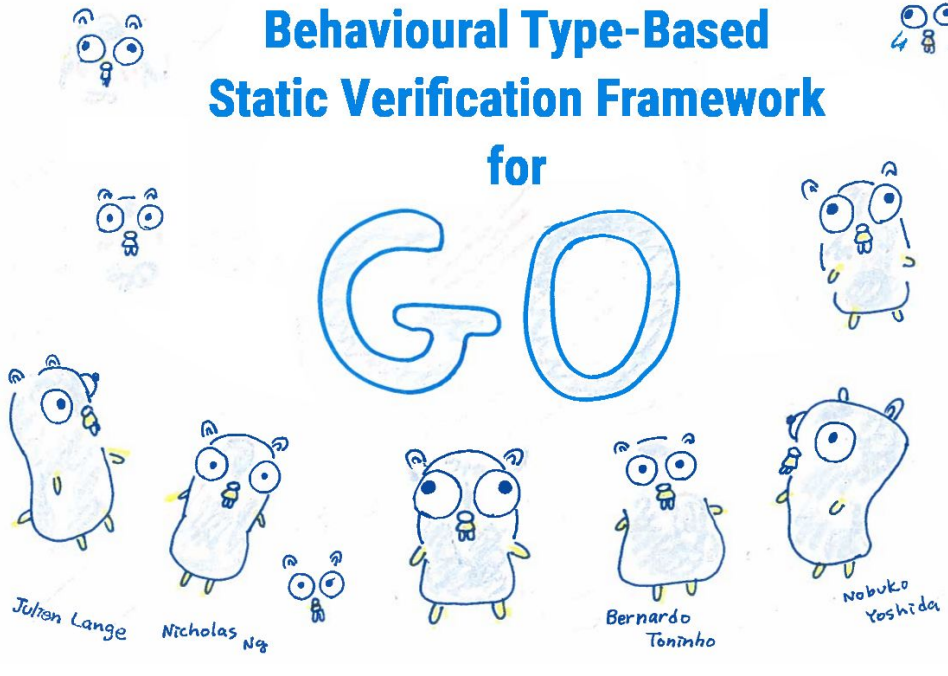
We present a library for the specification and implementation of distributed protocols in native F# (and other .NET languages) based on multiparty session types (MPST). There are two main contributions. Our library is the first practical development of MPST to support what we refer to as *interaction 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. 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 autocompletion. For example, an instantiation of the in-built type provider for WSDL Web services [6] may look like

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
hich
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

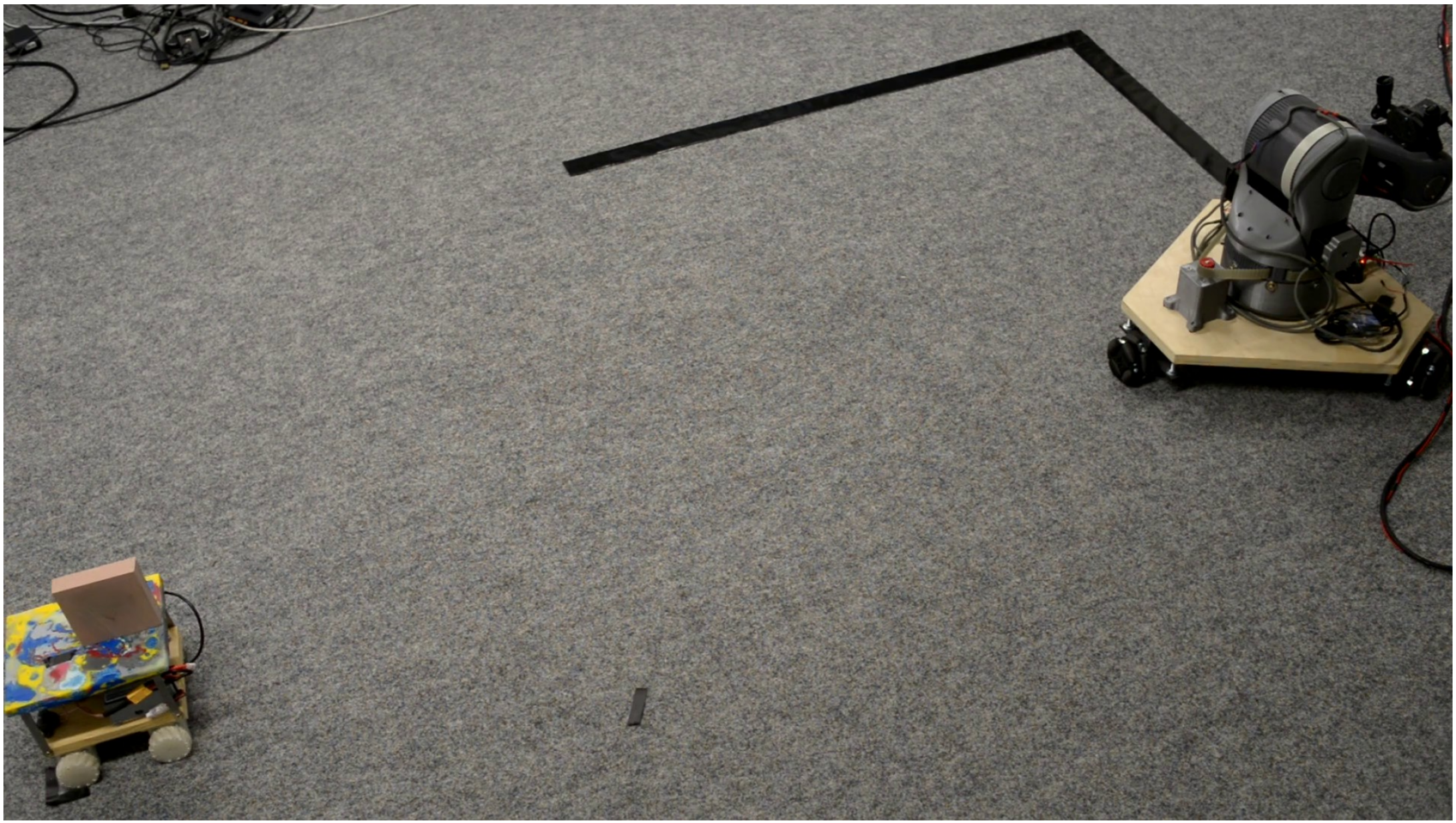
type and press enter

ARCHIVES

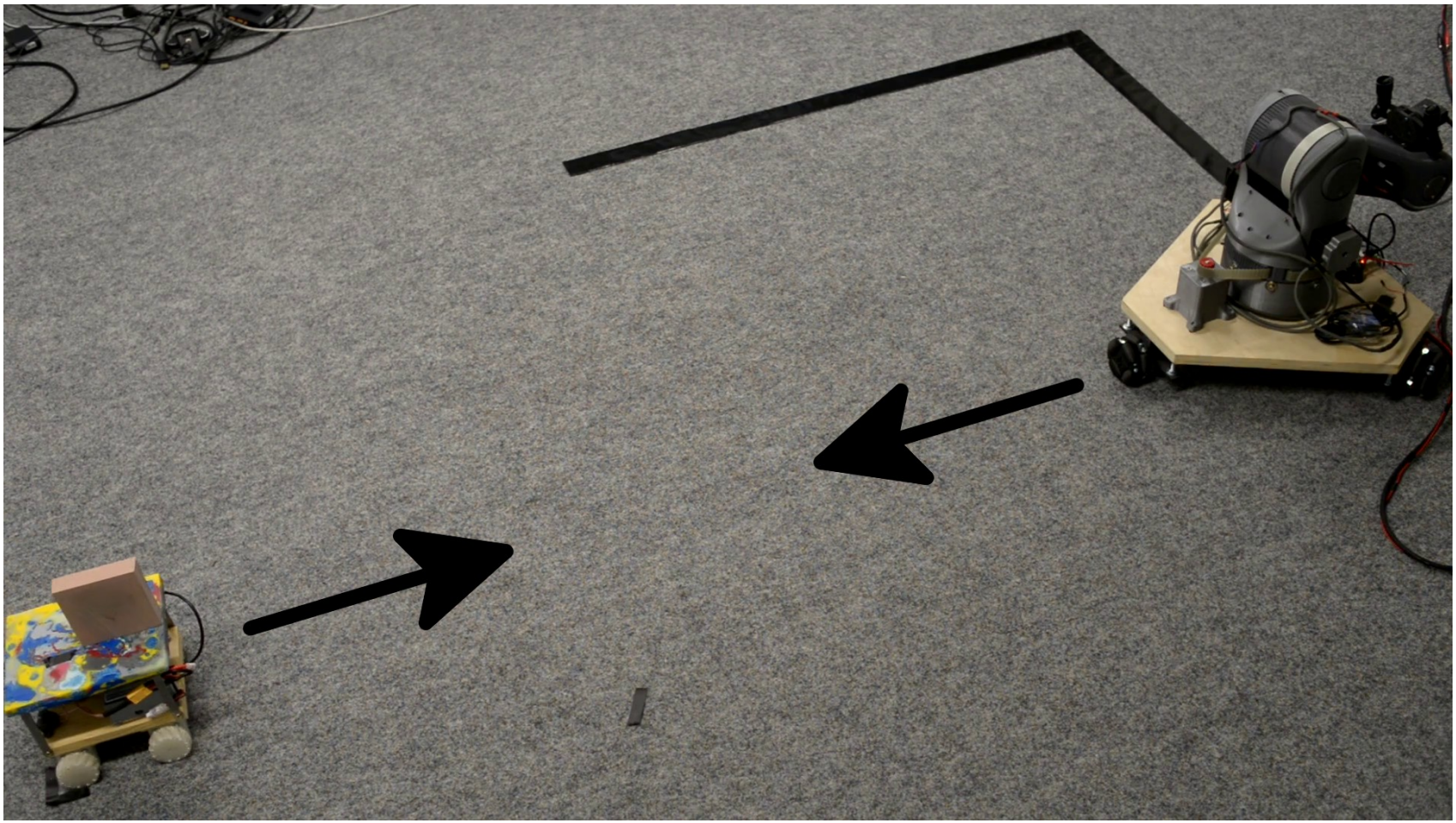
Select Month

MOST READ IN THE LAST FEW DAYS

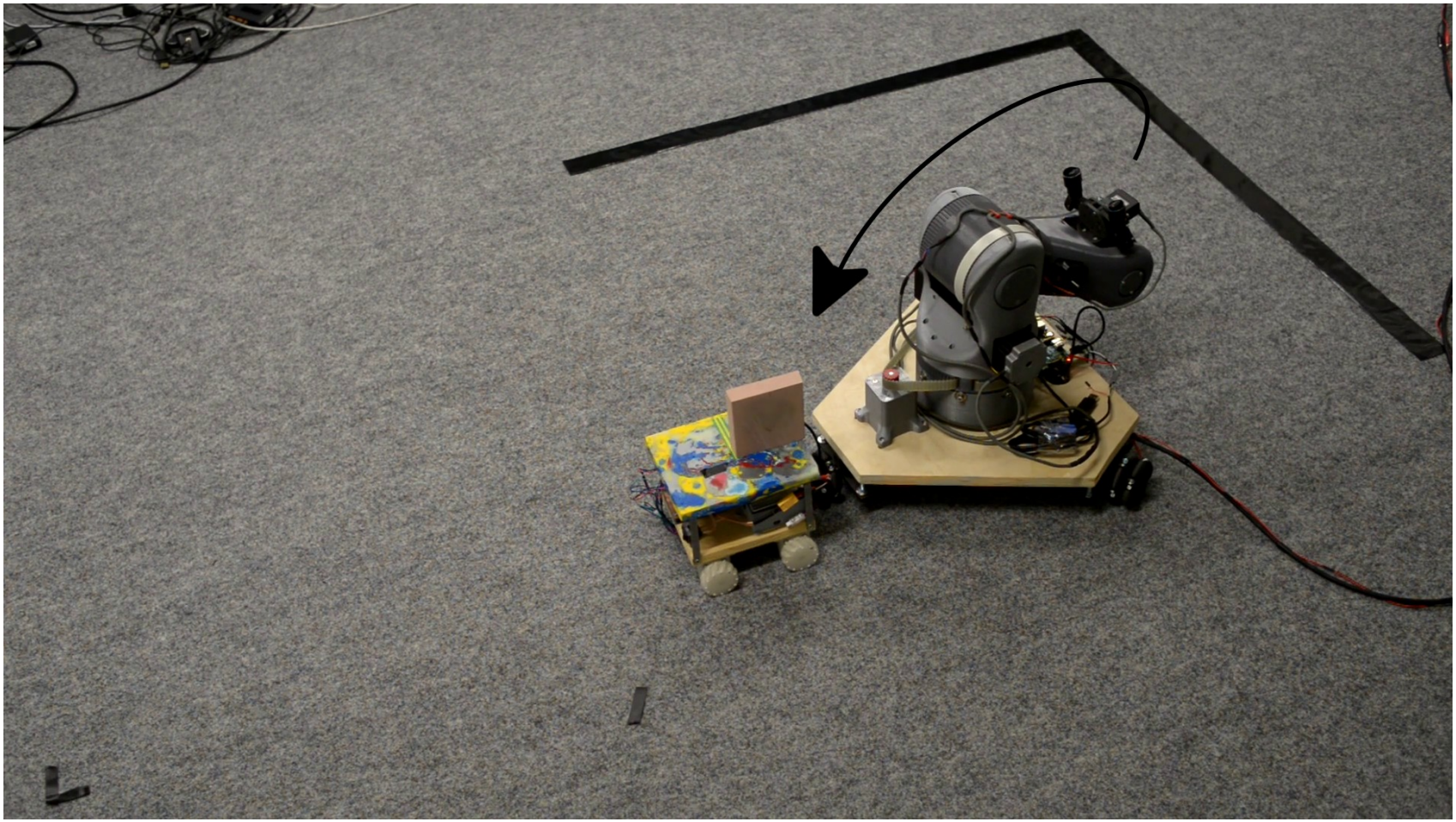
Impacts (V): Session Types for Robotics



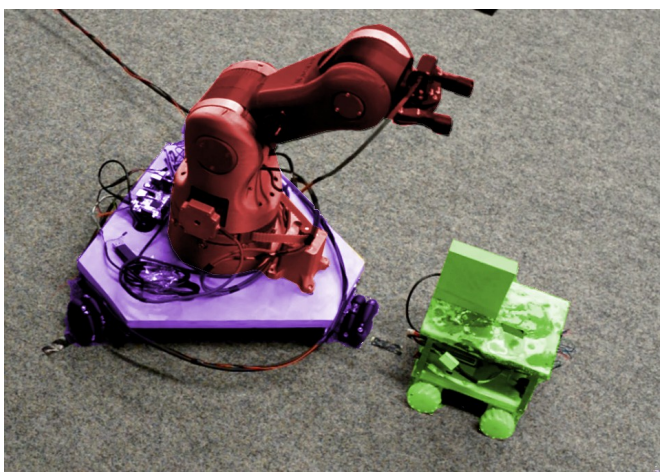
Impacts (V): Session Types for Robotics



Impacts (V): Session Types for Robotics



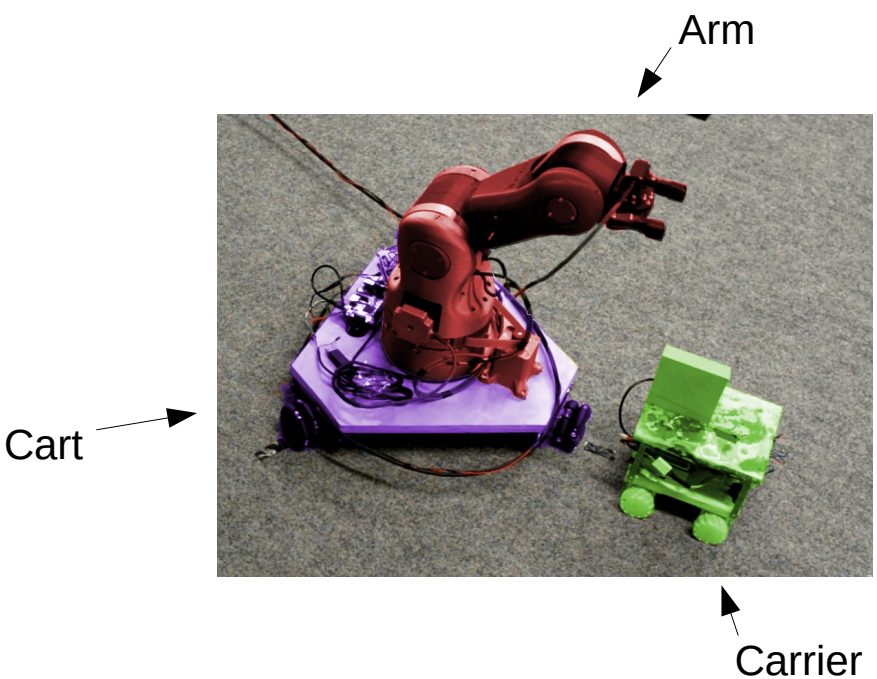
Impacts (V): Session Types for Robotics

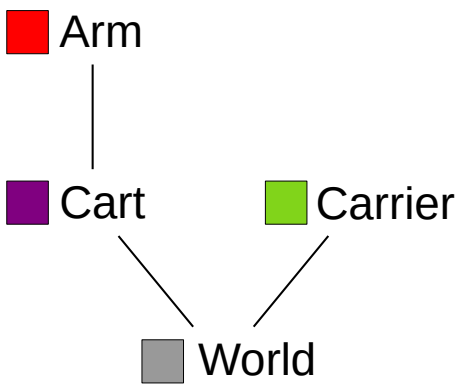


Cart & Arm:

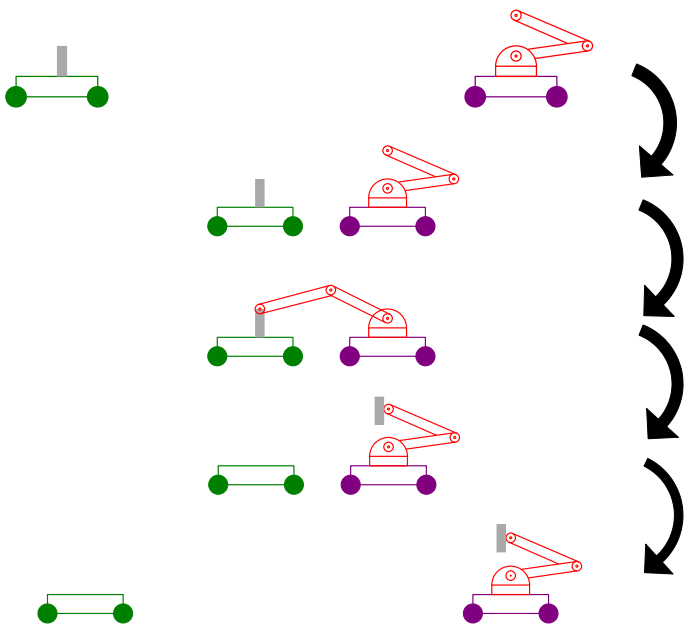
Two robots attached together that act as “one” robot (communication)

Impacts (V): Session Types for Robotics





Impacts (V): Session Types for Robotics



$\mathsf{Cart} \to \mathsf{Arm} : \mathit{fold}(\mathtt{unit}).\mathsf{dt}\langle \mathsf{Cart} : \mathsf{idle}, \mathsf{Carrier} : \mathsf{idle}, \mathsf{Arm} : \mathsf{fold}\rangle.$

$\mathsf{Arm} \to \mathsf{Cart} : \mathit{ok}(\mathtt{unit}).\mathsf{Cart} \to \mathsf{Carrier} : \mathit{ok}(\mathtt{unit}).$

$\mathsf{dt}\langle \mathsf{Cart} : \mathsf{move}, \mathsf{Carrier} : \mathsf{move}, \mathsf{Arm} : \mathsf{idle}\rangle.$

$\mathsf{Carrier} \to \mathsf{Cart} : \mathit{ok}(\mathtt{unit}).\mathsf{Cart} \to \mathsf{Arm} : \mathit{grab}(\mathtt{unit}).$

$\mathsf{dt}\langle \mathsf{Cart} : \mathsf{idle}, \mathsf{Carrier} : \mathsf{idle}, \mathsf{Arm} : \mathsf{grip}\rangle.$

$\mathsf{Arm} \to \mathsf{Cart} : \mathit{ok}(\mathtt{unit}).\mathsf{Cart} \to \mathsf{Carrier} : \mathit{ok}(\mathtt{unit}).$

$\mathsf{dt}\langle \mathsf{Cart} : \mathsf{move}, \mathsf{Carrier} : \mathsf{move}, \mathsf{Arm} : \mathsf{idle}\rangle.$

$\mathsf{Cart} \to \mathsf{Arm} : \mathit{done}(\mathtt{unit}).\mathsf{Cart} \to \mathsf{Carrier} : \mathit{done}(\mathtt{unit}).\mathtt{end}$

Selected Publications [2018-2019]

- [PLDI19] Alceste Scalas, NY, Elias Benussi: Verifying message-passing programs with dependent behavioural types.
- [CAV19] Julien Lange, NY: Verifying Asynchronous Interactions via Communicating Session Automata.
- [CONCUR19] Mario Bravetti, Marco Carbone, Julien Lange, NY, Gianluigi Zavattaro: A Sound Algorithm for Asynchronous Session Subtyping
- [FSE19] Nicola Atzei, Massimo Bartoletti, Stefano Lande, NY, Roberto Zunino:Developing secure Bitcoin contracts with BitML
- [ECOOP19] Rupak Majumdar, Marcus Pirron, NY, and Damien Zufferey, Motion Session Types for Robotic Interactions
- [FoSSaCs19] Simon Castellan, NY: Causality in Linear Logic
- [ESOP19] Laura Bocchi, Maurizio Murgia, Vasco T. Vasconcelos, NY: Asynchronous Timed Session Types
- [POPL19] Simon Castellan, NY: Two Sides of the Same Coin: Session Types and Game Semantics
- [POPL19] David Castro, Raymond Hu, Sung-Shik Jongmans, Nicholas Ng, NY: Distributed Programming Using Role Parametric Session Types in Go
- [POPL19] Alceste Scalas, Nobuko Yoshida: Less Is More: Multiparty Session Types Revisited
- [POPL19] Bernardo Toninho, NY: Interconnectability of Session Based Logical Processes
- [ICSE18] Julien Lange, Nicholas Ng, Bernardo Toninho, NY: A Static Verification Framework for Message Passing in Go using Behavioural Types
- [LICS18] Romain Demangeon, NY: Causal Computational Complexity of Distributed Processes
- [FoSSaCs18] Bernardo Toninho, Nobuko Yoshida: Depending On Session Typed Process
- [ESOP18] Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Tale of Two (Fully Abstract) Encodings
- [CC18] Rumyana Neykova, Raymond Hu, NY, Fahd Abdeljallal: A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#

Selected Publications [2018-2019]

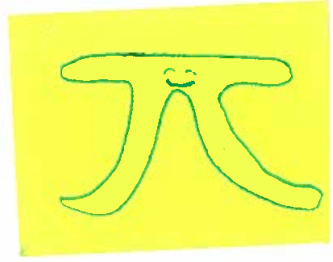
- [PLDI19] Alceste Scalas, NY, Elias Benussi: Verifying message-passing programs with dependent behavioural types.
- [CAV19] Julien Lange, NY: Verifying Asynchronous Interactions via Communicating Session Automata.
- [CONCUR19] Mario Bravetti, Marco Carbone, Julien Lange, NY, Gianluigi Zavattaro: A Sound Algorithm for Asynchronous Session Subtyping
- [FSE19] Nicola Atzei, Massimo Bartoletti, Stefano Lande, NY, Roberto Zunino:Developing secure Bitcoin contracts with BitML
- [ECOOP19] Rupak Majumdar, Marcus Pirron, NY, and Damien Zufferey, Motion Session Types for Robotic Interactions
- [FoSSaCs19] Simon Castellan, NY: Causality in Linear Logic
- [ESOP19] Laura Bocchi, Maurizio Murgia, Vasco T. Vasconcelos, NY: Asynchronous Timed Session Types
- [POPL19] Simon Castellan, NY: Two Sides of the Same Coin: Session Types and Game Semantics
- [POPL19] David Castro, Raymond Hu, Sung-Shik Jongmans, Nicholas Ng, NY: Distributed Programming Using Role Parametric Session Types in Go
- [POPL19] Alceste Scalas, Nobuko Yoshida: Less Is More: Multiparty Session Types Revisited
- [POPL19] Bernardo Toninho, NY: Interconnectability of Session Based Logical Processes
- [ICSE18] Julien Lange, Nicholas Ng, Bernardo Toninho, NY: A Static Verification Framework for Message Passing in Go using Behavioural Types
- [LICS18] Romain Demangeon, NY: Causal Computational Complexity of Distributed Processes
- [FoSSaCs18] Bernardo Toninho, Nobuko Yoshida: Depending On Session Typed Process
- [ESOP18] Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Tale of Two (Fully Abstract) Encodings
- [CC18] Rumyana Neykova, Raymond Hu, NY, Fahd Abdeljallal: A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#

on Polymorphic Sessions and Functions or a Tale of Two Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

On Polymorphic Sessions and Functions

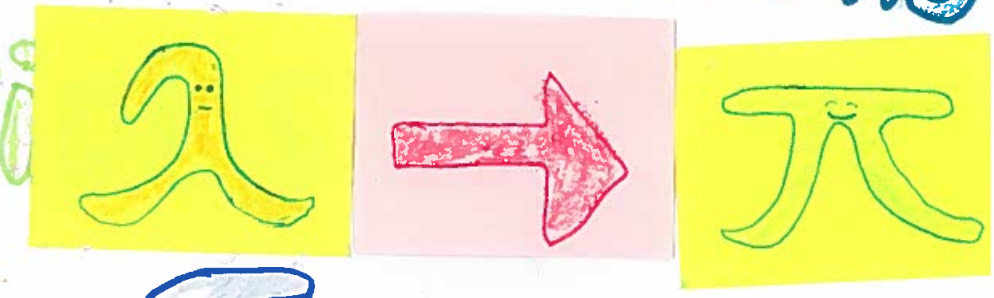


A Tale of Two Fully Abstract Encodings

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

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

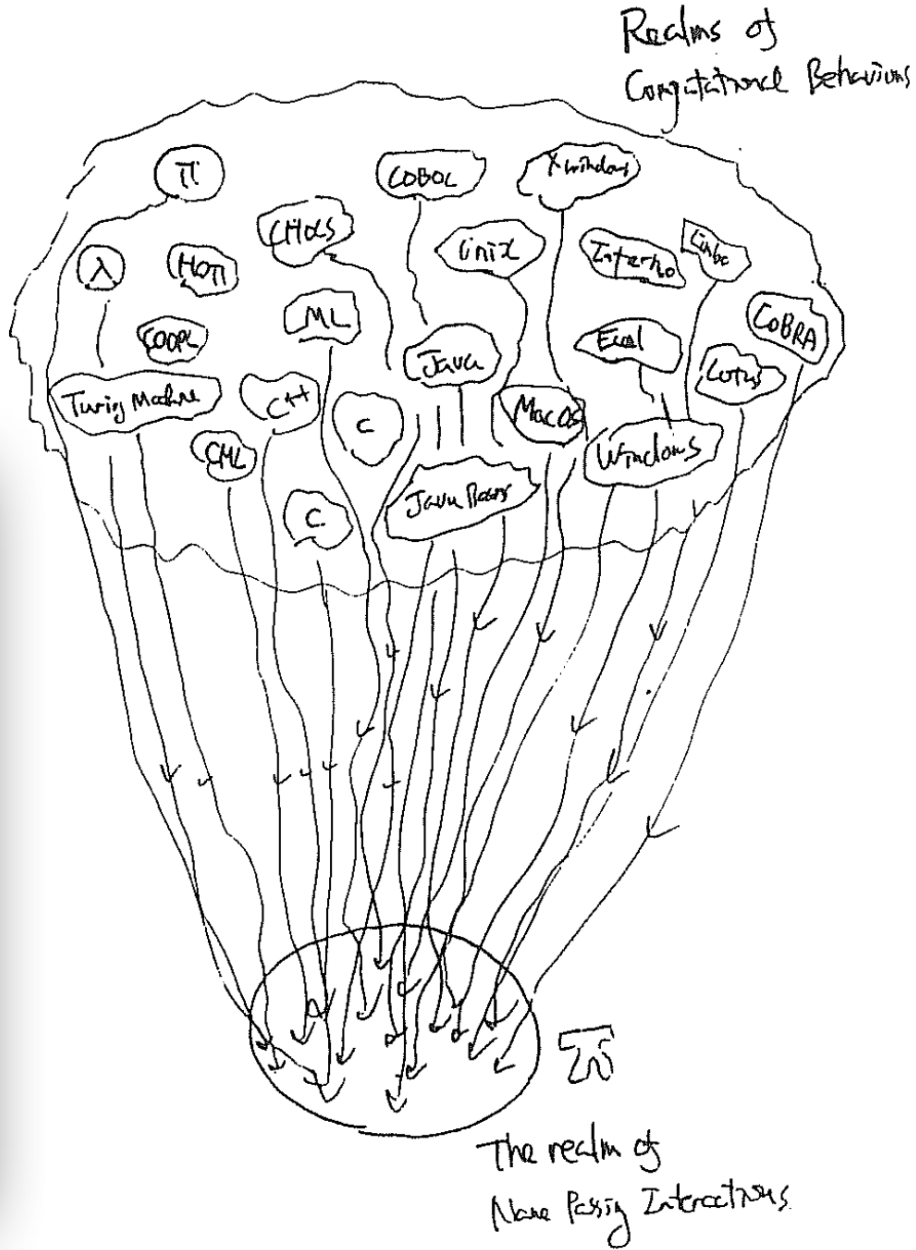
$[\![\lambda x.M]\!]_u \stackrel{\text{def}}{=} u(xu').[\![M]\!]_{u'}$

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

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

* Examples of Representable Computation.

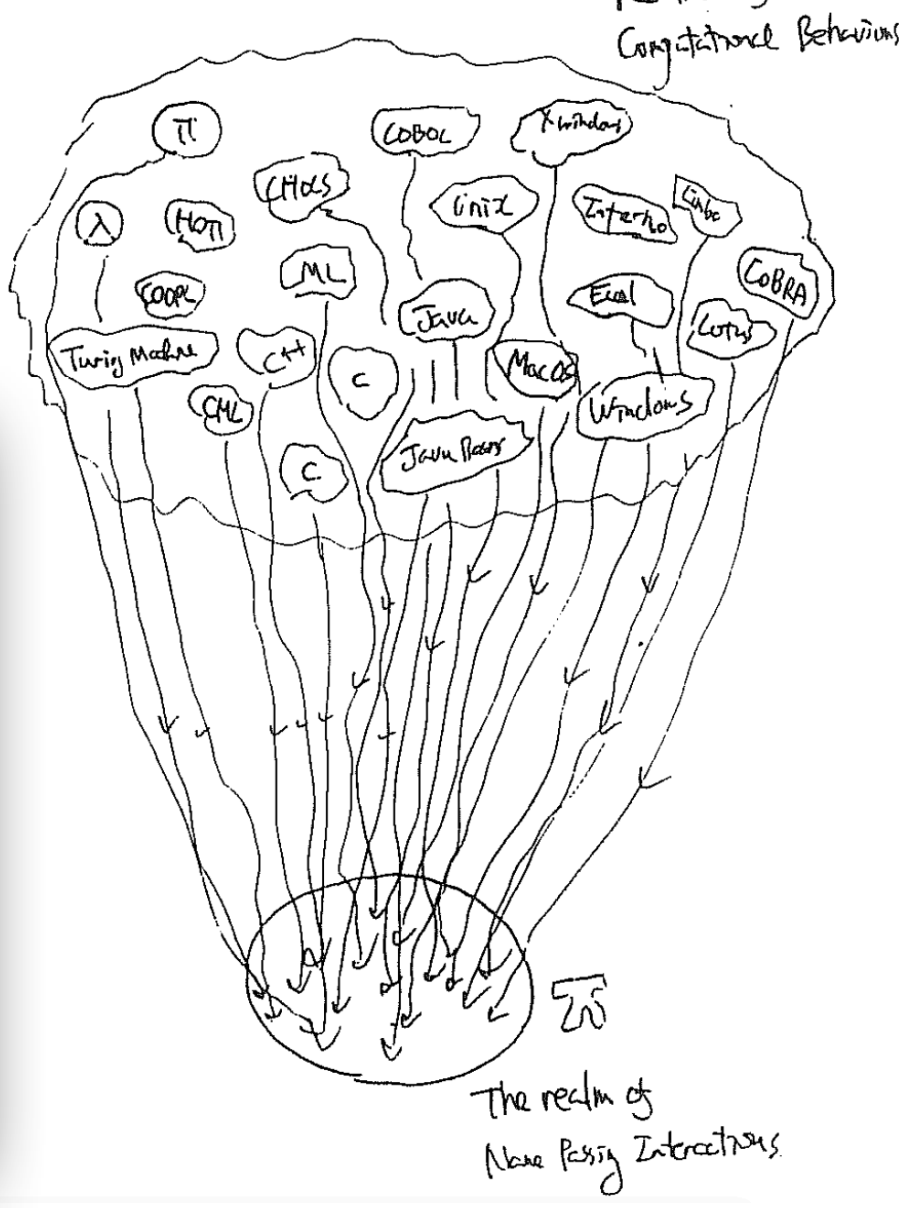
- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner92, ...]
- Proof Nets [Bellin and Scott93]
- Arbitrary 'constant' interaction [HYS4]
- strategies on Games [HO95]

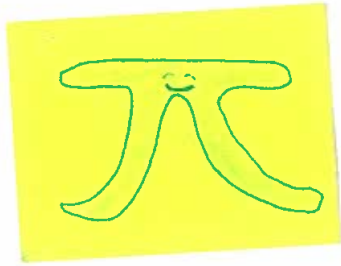


* Examples of Representable Computation.

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]
- ⋮



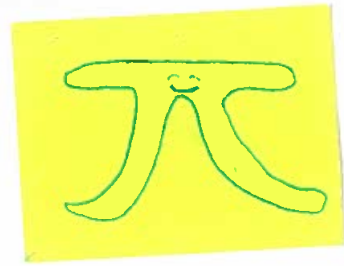


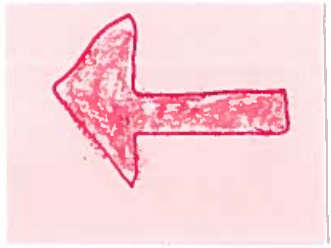
$M \simeq N$

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

Conextual Congruence

Contextual Congruence



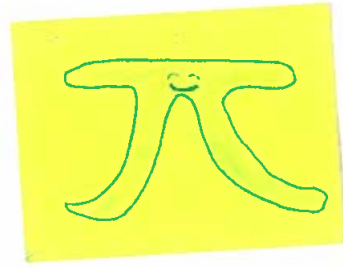
$M \simeq N$  $[\![M]\!] \simeq [\![N]\!]$

Conextual Congruence

Contextual Congruence

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

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

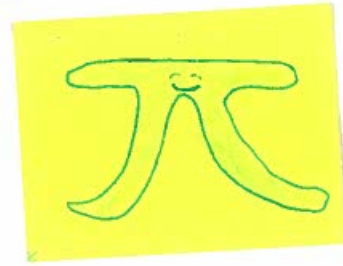


$M \simeq N$ 

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

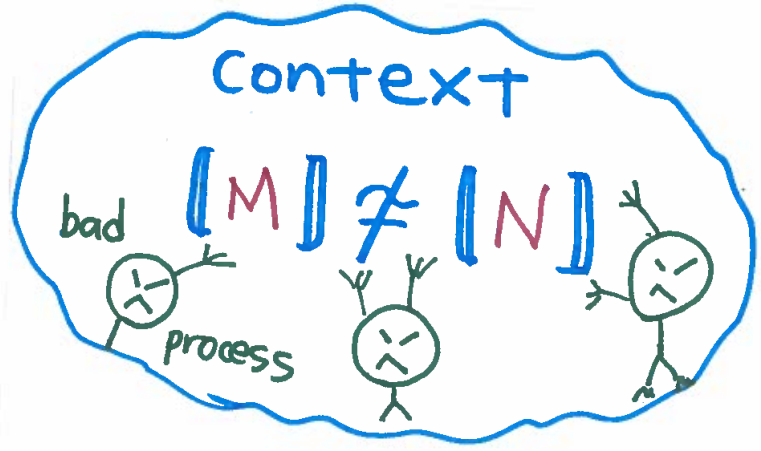
$C[P]\Downarrow_a$ iff $C[Q]\Downarrow_a$

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



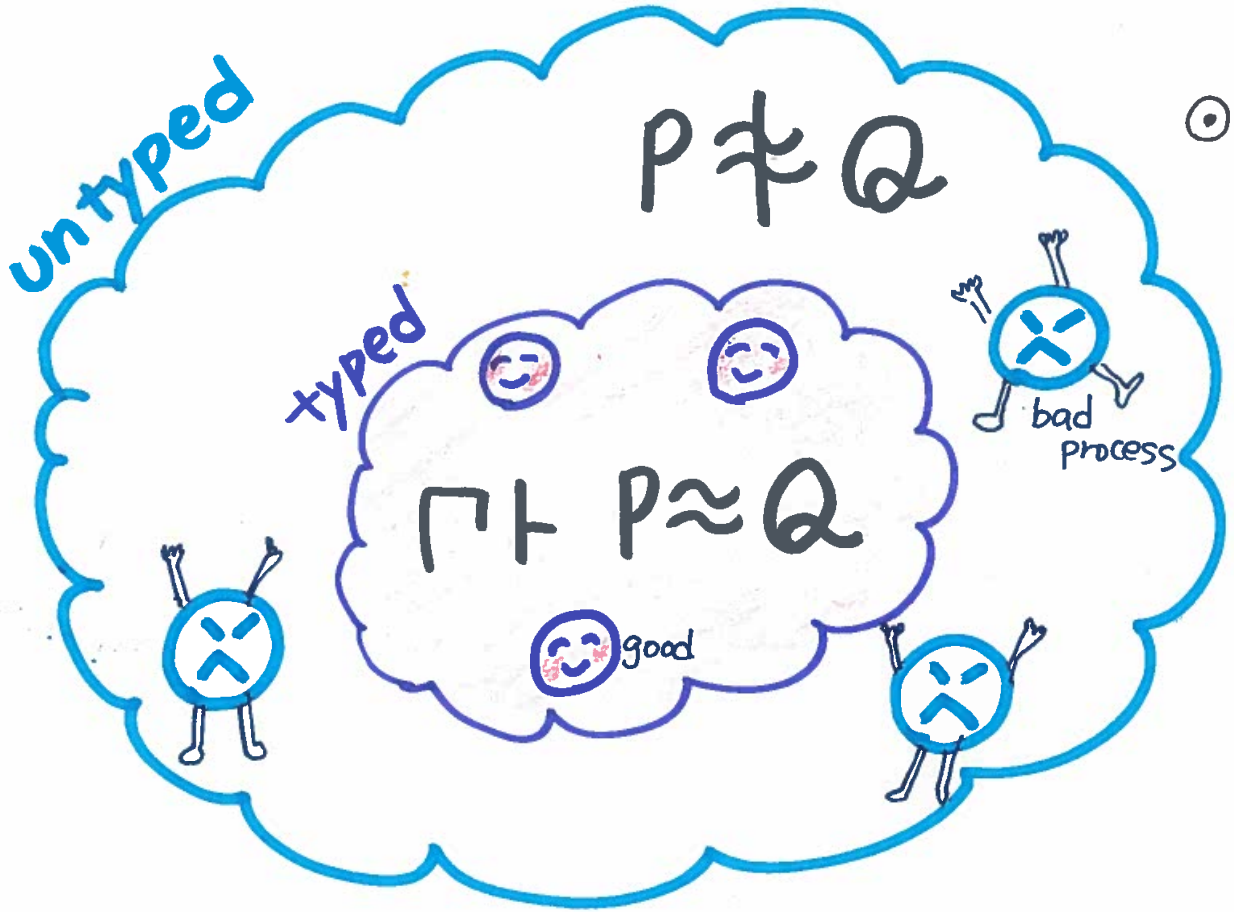
$M \simeq N$ 

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



Typed Semantics in π 1991→

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



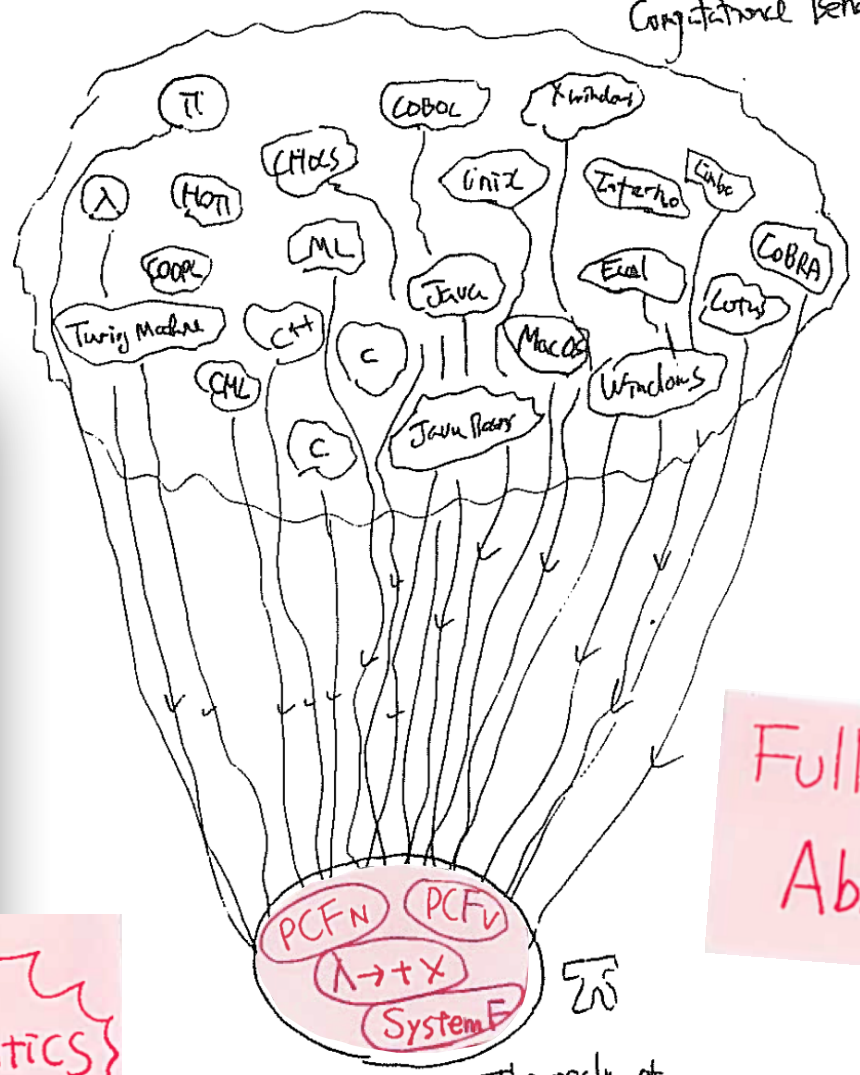
- ⊙ Correctness of Encoding
- ⊙ Limit environment Γ
  ⇒ Equate more processes
- ⊙ Compositional

* 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

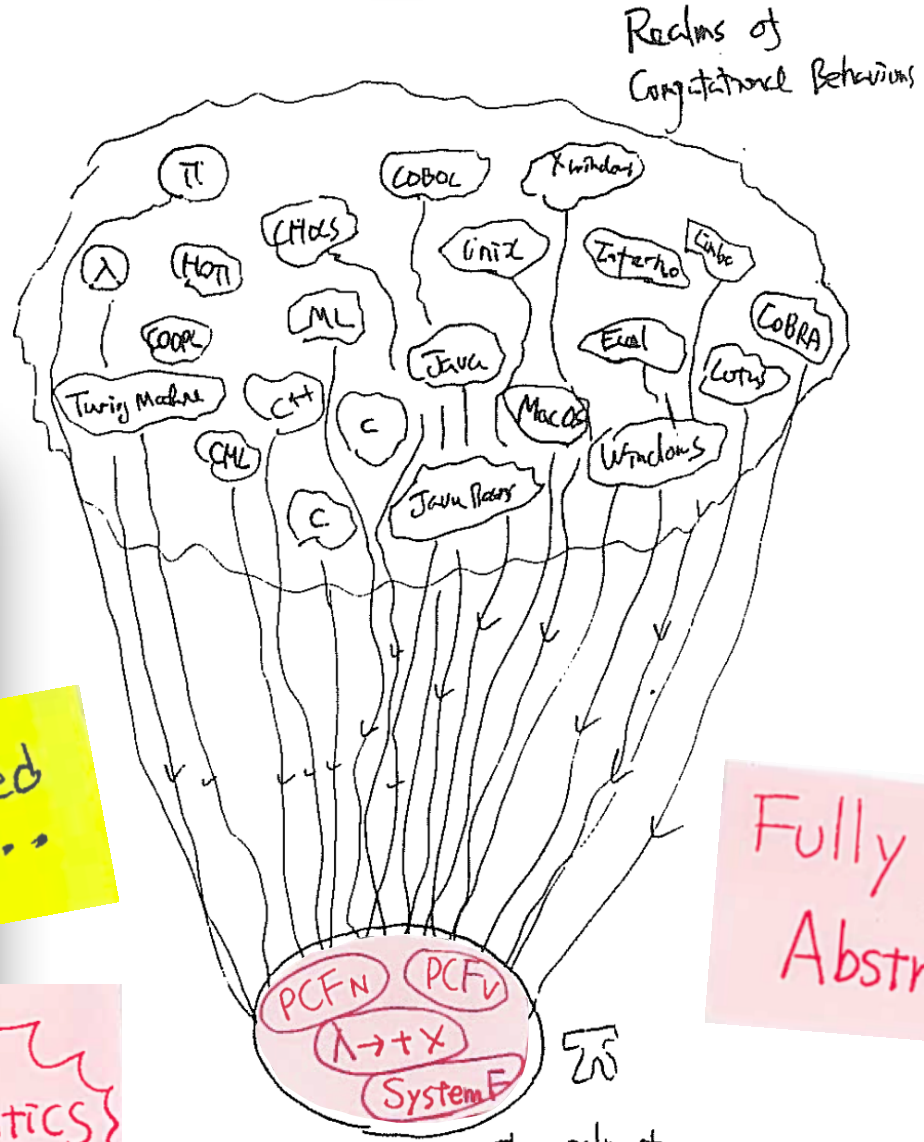
Fully Abstract

* Examples of Representable Computation.

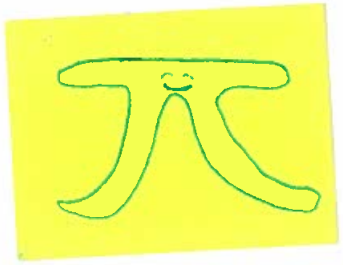
- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [
- Strategies on Games [HO95]

Complicated ...

Game Semantics

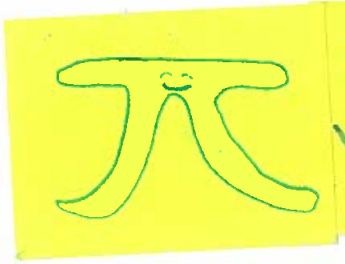


Fully Abstract



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

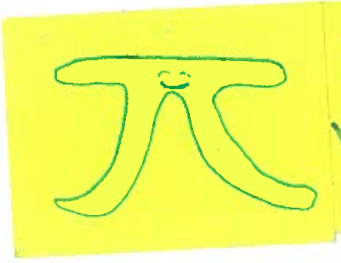
System F



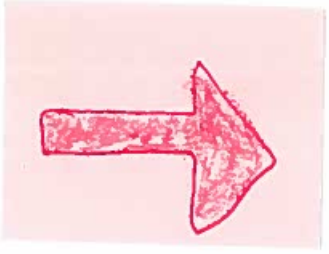
Session $\Pi\Pi$

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

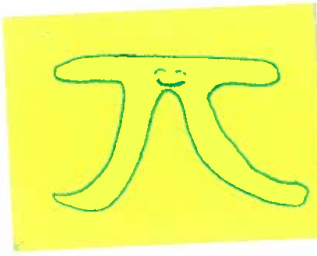
System F



Session

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

System F



Session Π

$M \simeq N$

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

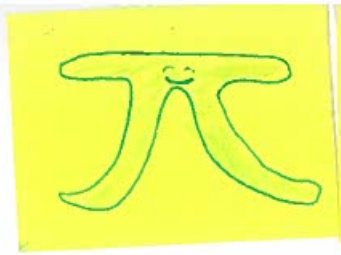
$[\![P]\!] \simeq [\![Q]\!]$

$P \simeq Q$

Reverse

New

System F



Session

$M \simeq N$

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

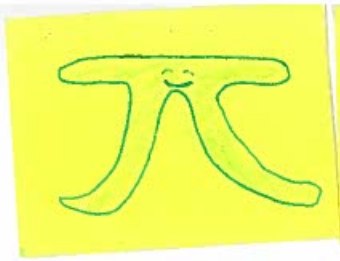
$[\![P]\!] \simeq [\![Q]\!]$

$P \simeq Q$

Reverse

New

System F



Session $\Pi\!\!\Pi$

$M \simeq N$

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

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

$P \simeq Q$

TWO APPLICATIONS

Session π

Caires, Pérez, Pfenning, Toninho 13

Types

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

Processes

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

Judgement

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

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

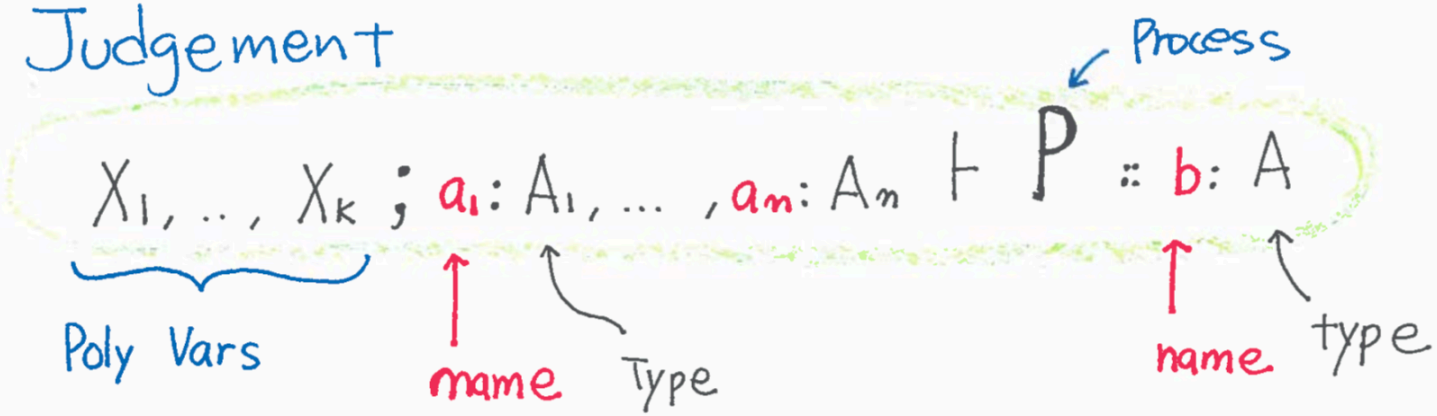
Type

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

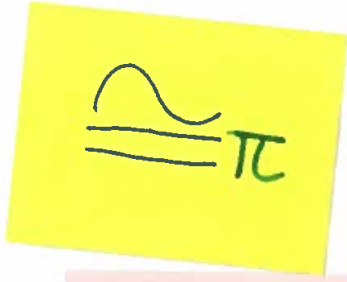
Polymorphic Session

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

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

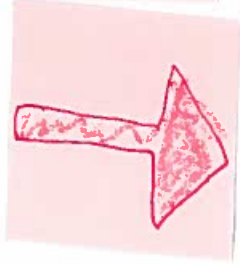
$$[\![(\!|\, P \,|\!)]\!]_z \cong P$$

Full Abstraction

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

$$M \cong N$$

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

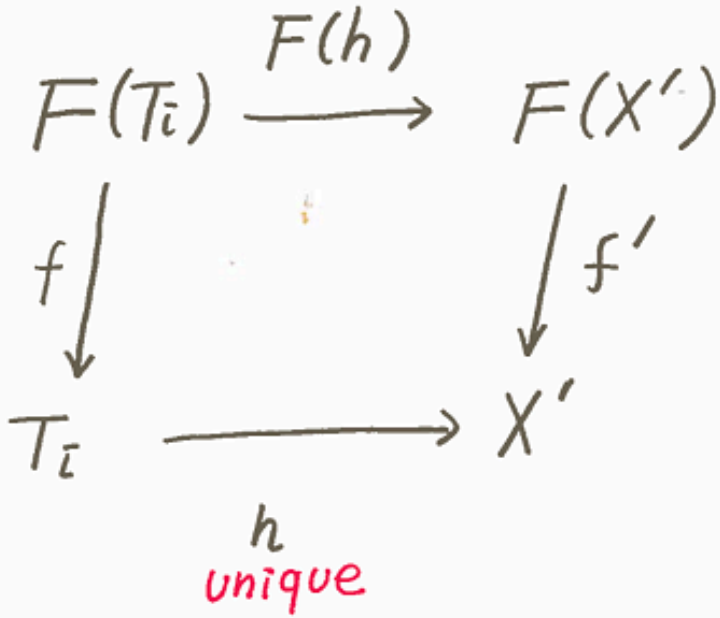


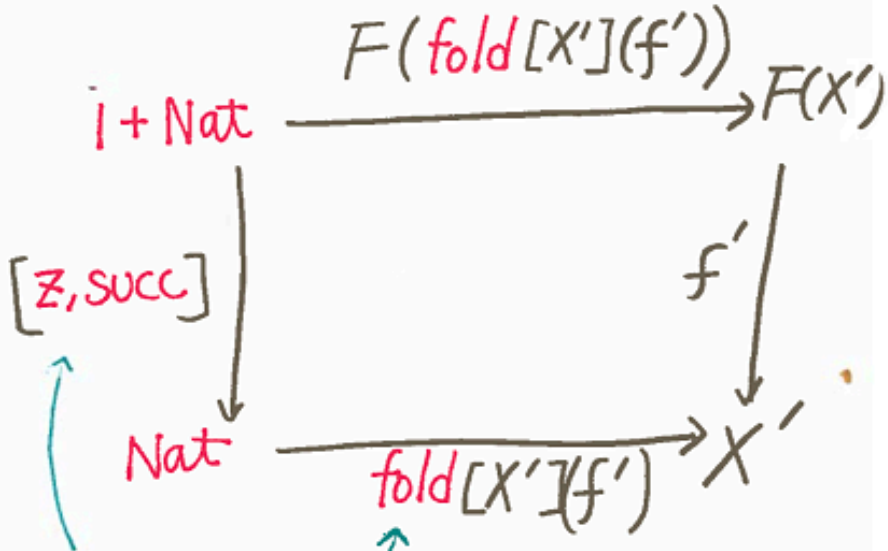
$$P \cong Q$$

Application 1 Inductive and Coinductive Types

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

as Initial / Final (Co)Algebra

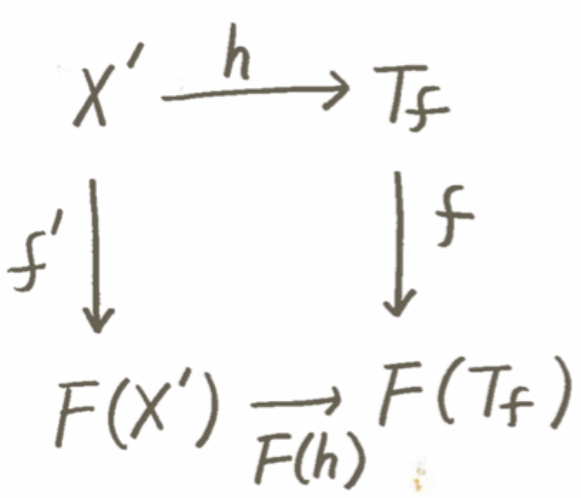


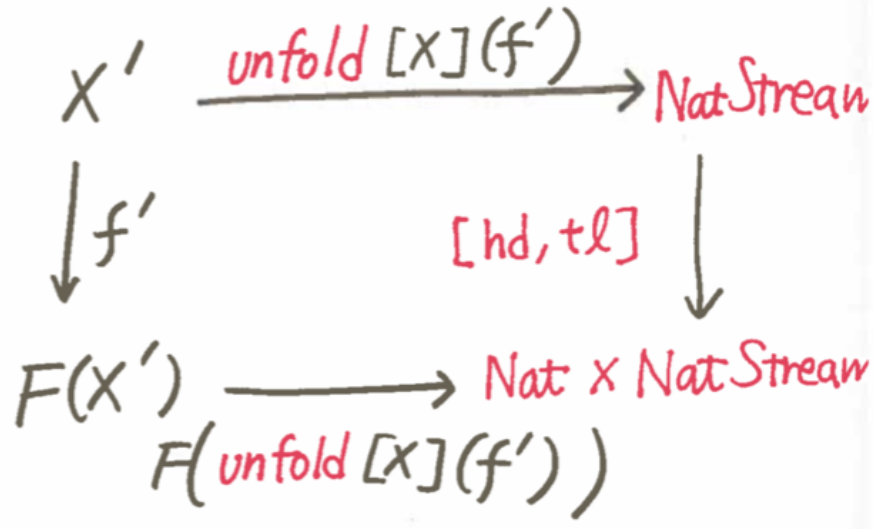


can be defined in System F and fold is unique for all inductive types

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

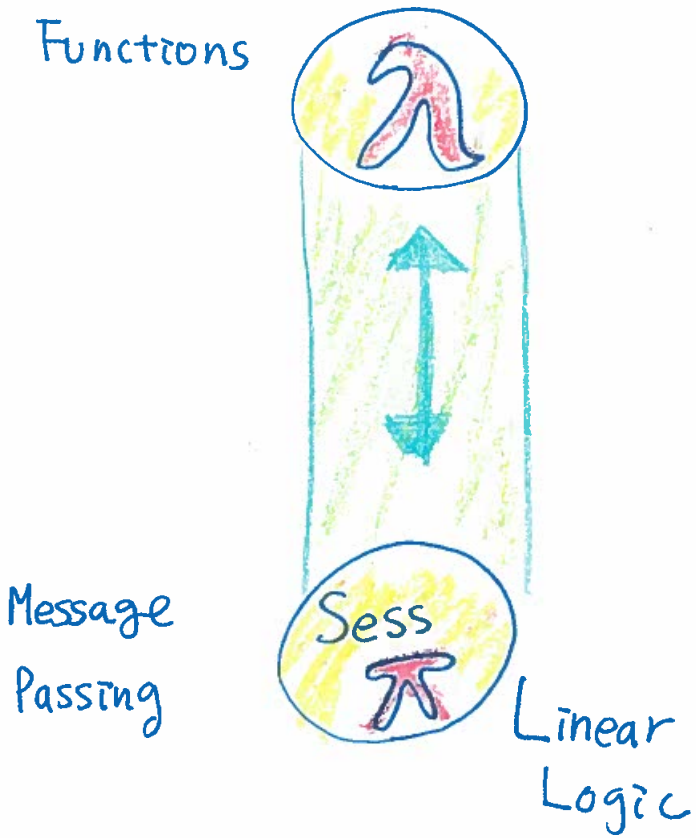
- Full Abstraction / Inverse $x\langle M\rangle.P$
- Strong Normalisation via Strong Normalisation

HOπ : λ

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

cf. [CPPT'13] logical relation

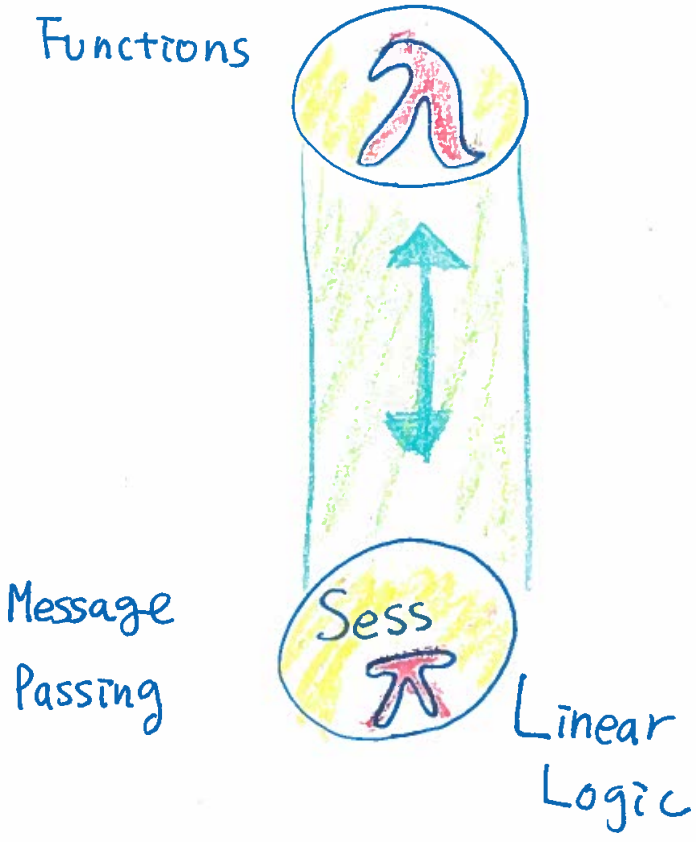
Summary



LL-based

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

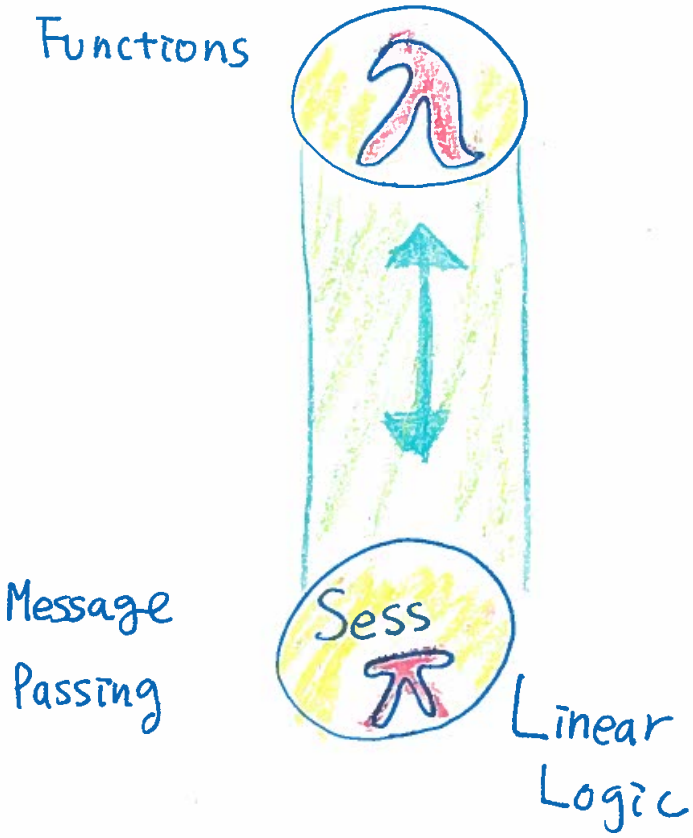
Summary



LL-based

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

Summary



LL-based

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