

on Polymorphic
and Sessions
Functions

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/>

SESSION TYPES
 π MobilityReadingGroup
 π -calculus, Session Types research at Imperial College

Home People Publications Grants Talks Tutorials Tools Awards Kohei Honda

NEWS

The paper 'Language primitives and type discipline for structured communication-based programming' by Kohei Honda, Vasco T. Vasconcelos and Makoto Kubo, has received the ETAPS 2019 Test-of-Time Award.

» more

11 Apr 2019

The paper 'A fully abstract game semantics for general references' by Samson Abramsky, Kohei Honda and Guy McCusker, has received the LICS Test-of-Time Award.

SELECTED PUBLICATIONS

2019

Simon Castellan, Nobuko Yoshida: [Causality in Linear Logic](#). FoSSaCS 2019 : 150 - 168.

David Castro, Raymond Hu, Sung-Shik Jongmans, Nicholas Ng, Nobuko Yoshida: [Distributed Programming Using Role Parametric Session Types in Go](#). POPL 2019 : 29:1 - 29:30.

Tiago Cogumbreiro, Raymond Hu, Francisco Martins, Nobuko Yoshida: [Dynamic deadlock verification for general barrier synchronisation](#). TOPLAS : 1 - 38.



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



SIGPLAN

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.

Impacts (I): Scribble



www.scribble.org

The screenshot shows the homepage of the Scribble project. At the top, there is a navigation bar with links for Home, Getting Started, Downloads, Documentation, and Community. Below this is a large blue header with the text "Scribble: Describing Multi Party Protocols". The main content area features a paragraph explaining that Scribble is a language for describing application-level protocols and that it enables verification to prevent unintended consequences like deadlocks. Below the paragraph are five columns, each with a title, an icon, and a brief description of a feature or concept.

Home Getting Started Downloads Documentation - Community -

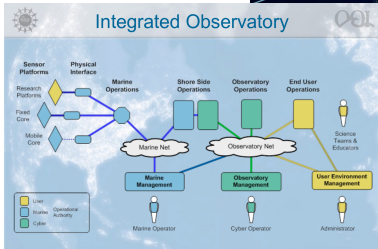
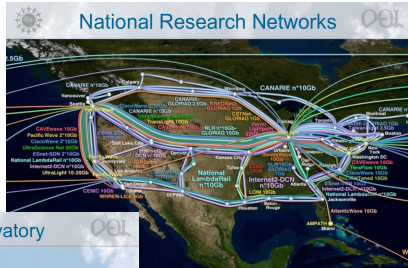
Scribble: Describing Multi Party Protocols

Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.

- Describe** Scribble is a language for describing multiparty protocols from a global, or endpoint neutral, perspective.
- Verify** Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and do not suffer from deadlocks or livelocks.
- Project** Endpoint projection is the term used for identifying the responsibility of a particular role (or endpoint) within a protocol.
- Implement** Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c) statically verify the code against the projection.
- Monitor** Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of a particular endpoint, to ensure it correctly implements the expected behaviour.

Impacts (II): Ocean Observatories Initiative

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



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



Estafet

Innovate | Deliver | Transform

1. All design work takes place in ABACUS, DCC's enterprise architecture tool. This can export standard XMI files (an open standard for UML5)

2. XMI is converted into OpenTracing format for consumption by managed service



OPENTRACING



3. OpenTracing files are combined to build a model in Scribble

4. Model holds *types* rather than *instances* to understand behaviour

5. Scribble compiler identifies inconsistency, change & design flaws

6. Issues highlighted graphically in Eclipse

www.estafet.com

Estafet Managed Service



7. Generate exception report and send back to DCC

End-to-End Switching Programme by DCC



Estafet

Innovate | Deliver | Transform

Caveats:

1. Using earlier implementation of Scribble (CDL), because we already have those tools
2. Using earlier plugin to Eclipse - we'd want to improve this
3. We're not going via OpenTracing - this is part of the bid costs



7. Generate exception report and send back to DCC

Scope of the demo



OPENTRACING

3. OpenTracing files are combined to build a model in Scribble



4. Model holds *types* rather than *instances* to understand behaviour



5. Scribble compiler identifies inconsistency, change & design flaws



6. Issues highlighted graphically in Eclipse

www.estafet.com

Estafet Managed Service

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 on 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 auto-completion. For example, an instantiation of the in-built type provider for WSDL Web services [6] may look like



Graydon Hoare

@graydon_pub

(This stuff is _fantastic_)

11:31 PM - 11 Mar 2018

32 Retweets 83 Likes



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



Juhen Lange



Nicholas Ng



Bernardo Toninho



Nobuko Yoshida



GOLANG UK CONFERENCE

16th*, 17th & 18th AUGUST 2017

The Brewery, London



Imperial College
London

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

Go concurrency verification research at DoC grabs headline

POPL'17

the morning paper

ICSE'18

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

Home About Info 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

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

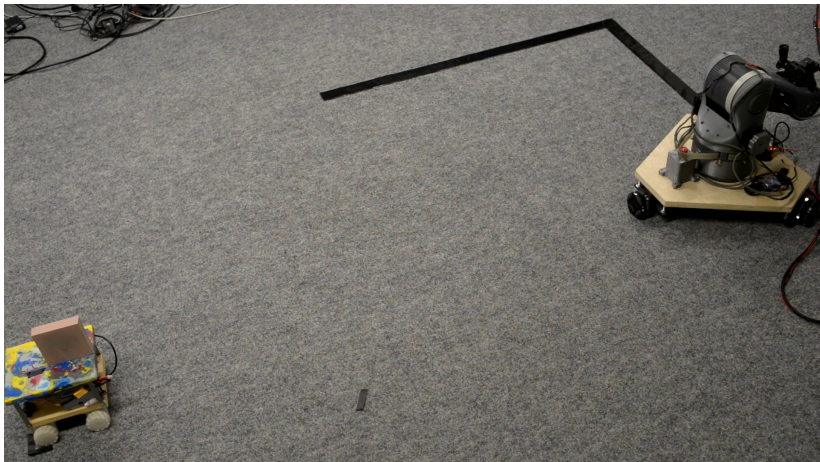
Select Month

MOST READ IN THE
LAST FEW DAYS

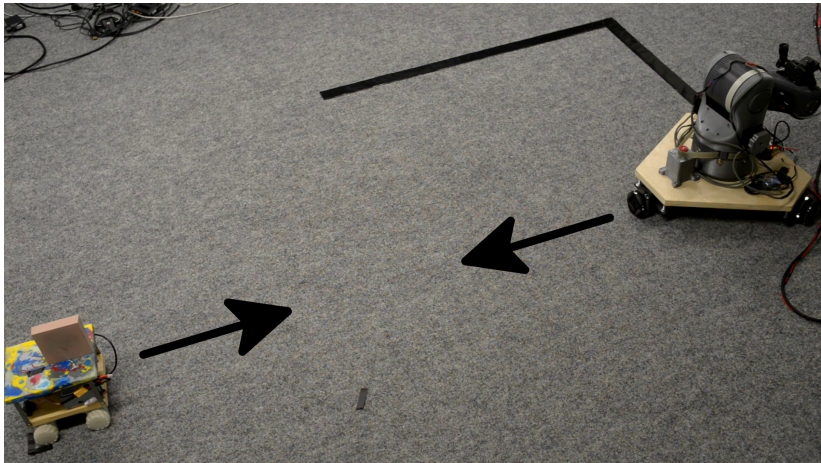
currency
rates a

tured in the
'high
interesting
easily
le of the
L (Principles

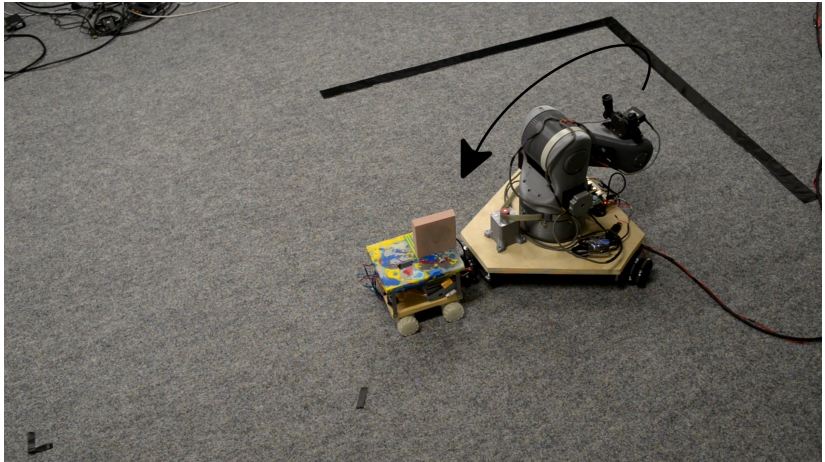
Impacts (V): Session Types for Robotics



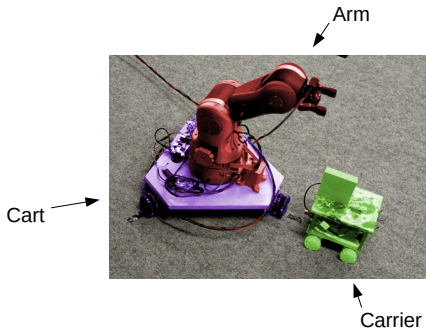
Impacts (V): Session Types for Robotics



Impacts (V): Session Types for Robotics

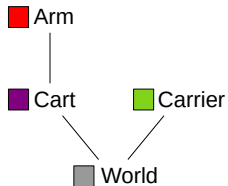
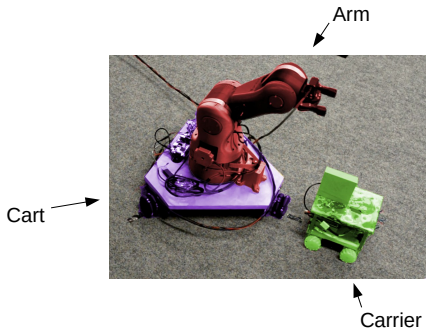


Impacts (V): Session Types for Robotics

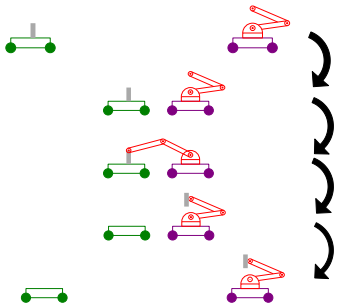


Cart & Arm:
Two robots attached together that
act as “one” robot (communication)

Impacts (V): Session Types for Robotics



Impacts (V): Session Types for Robotics



```
Cart → Arm : fold(unit).dt(Cart : idle, Carrier : idle, Arm : fold).  
Arm → Cart : ok(unit).Cart → Carrier : ok(unit).  
dt(Cart : move, Carrier : move, Arm : idle).  
Carrier → Cart : ok(unit).Cart → Arm : grab(unit).  
dt(Cart : idle, Carrier : idle, Arm : grip).  
Arm → Cart : ok(unit).Cart → Carrier : ok(unit).  
dt(Cart : move, Carrier : move, Arm : idle).  
Cart → Arm : done(unit).Cart → Carrier : done(unit).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] [Ramyana Nayana](#), [Raymond Hu](#), NY, [Fahd Abdeljallal](#): [A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#](#)

on Polymorphic
and Sessions
Functions

a Tale of Two
Encodings

Bernardo Toninho @Nova

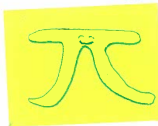
Nobuko Yoshida @Imperial

on Polymorphic

Sessions

and

Function



a Tale of Two

Fully
Abstract

Encodings

Bernardo Toninho @ Nova

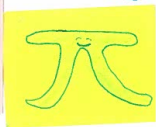
Nobuko Yoshida @ Imperial

on Polymorphic

Sessions

and

Function



a Tale of Two

Fully
Abstract

Encodings

Bernardo Toninho @ Nova

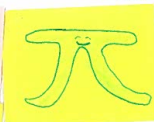
Nobuko Yoshida @ Imperial

on Polymorphic

Sessions

and

Function



a Tale of Two

Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

The π -calculus as a Descriptive Tool

by Kohei
Honda
1995

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

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

$$\text{with } \pi ::= x \langle \bar{a} \rangle \mid \bar{x} \langle a \rangle.$$

Milner's
Encoding
1991

λ in π

$$[x]_u \triangleq \bar{x}(u).$$

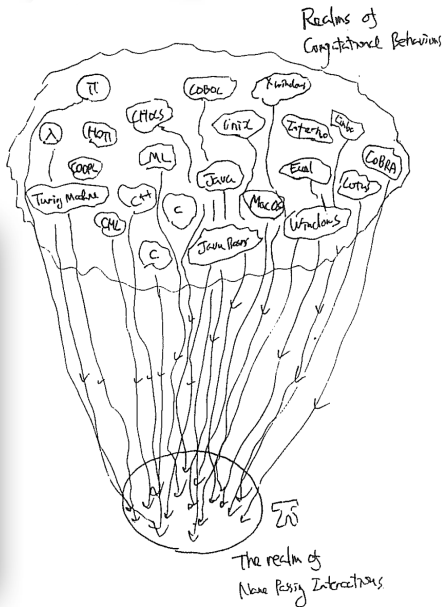
$$[\lambda x.M]_u \triangleq u(x).[M]_u.$$

$$[MN]_u \triangleq (vz) ([M]_z \mid \bar{z}(u) \mid [N]_u)$$

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

* Examples of Representable Computation.

- λ -calculus [MPW89, Milner 90, Milner 92, ...]
 - Concurrent Object [Walker 81]
 - ω -order term passing [Stangor 92]
 - Various data structures [Milner 92, ...]
 - Proof Nets [Bellare and Scott 93]
 - Abstract 'constant' interaction [HY94]
 - Strategies on Games [HO95]
- ⋮



* Examples of Representable Computation.

Operationally

Sound

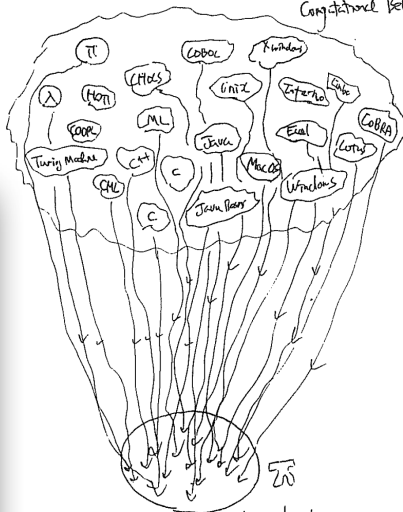
but NOT

Fully Abstract

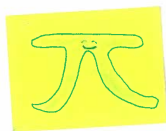
- λ -calculus [MPW89, Milner 90, Milner 92, ...]
- Concurrent Object [Walker 81]
- ω -order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellare and Scott 93]
- Abstract 'constant' interaction [HY94]
- Strategies on Games [HO95]

⋮

Realms of Computational Behaviors



The realm of Name Passing Interactions



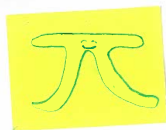
$M \approx N$



$\llbracket M \rrbracket \approx \llbracket N \rrbracket$

Contextual
Congruence

Contextual
Congruence



$M \approx N$



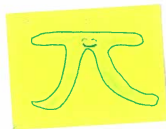
$\llbracket M \rrbracket \approx \llbracket N \rrbracket$

Contextual
Congruence

Contextual
Congruence

$C[P] \Downarrow_a \text{ iff } C[Q] \Downarrow_a$

$C[\llbracket M \rrbracket] \Downarrow_a \text{ iff } C[\llbracket N \rrbracket] \Downarrow_a$



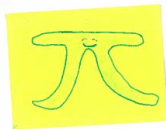
$M \approx N$



$\llbracket M \rrbracket \approx \llbracket N \rrbracket$

$C[P] \Downarrow_a \text{ iff } C[Q] \Downarrow_a$

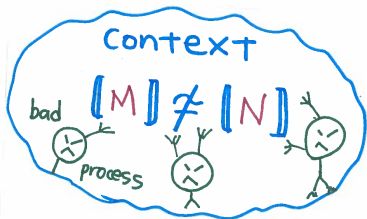
$C[\llbracket M \rrbracket] \Downarrow_a \text{ iff } C[\llbracket N \rrbracket] \Downarrow_a$



$$M \approx N$$

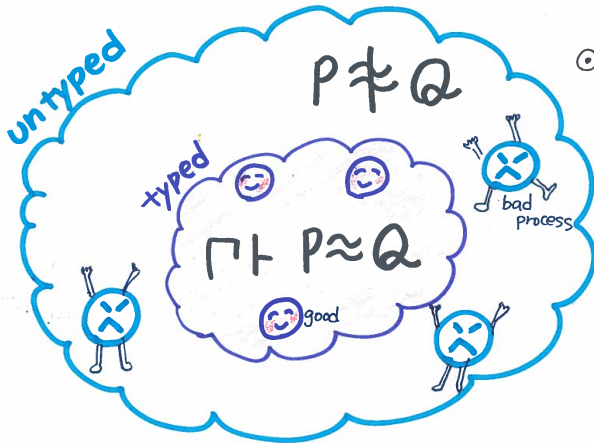


$$[M] \approx [N]$$



Typed Semantics in π 1991 \rightarrow

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

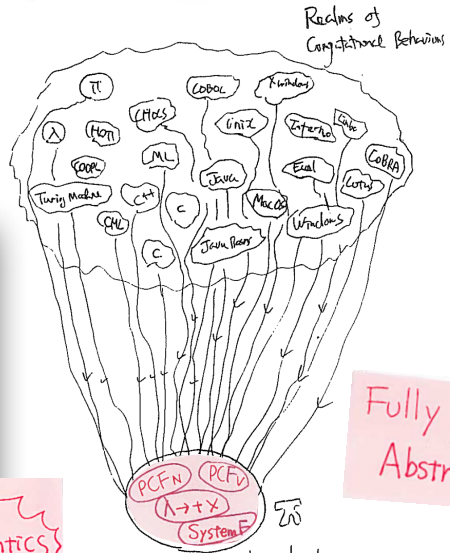


- ⊙ Correctness of Encoding \sqsupset
- ⊙ Limit environment \sqsupset
 \Rightarrow Equate more processes
- ⊙ Compositional

* Examples of Representable Computation.

- λ -calculus [MPW89, Milner 90, Milner 92, ...]
- Concurrent Object [Walker 81]
- ω -order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellare and Scott 93]
- Abstract 'constant' interaction [HY94]
- Strategies on Games [HO95]
- ...

Game Semantics



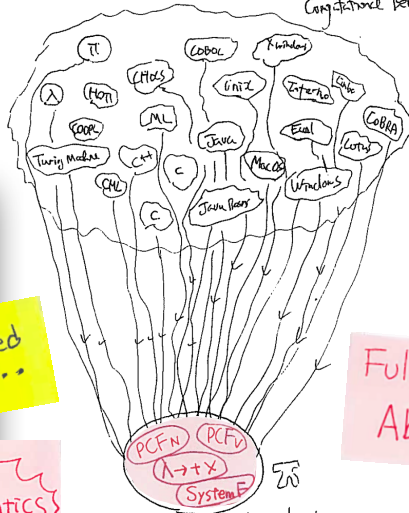
Realms of Computational Behaviors

Fully Abstract

The realm of Name Passing Interactions

* Examples of Representable Computation.

Realms of Computational Behaviors



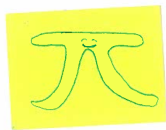
- λ -calculus [MPW83, Milner 90, Milner 92, ...]
- Concurrent Object [Walker 81]
- ω -order term passing [Sangiorgi 92]
- Various data structures [Milner 81]
- Proof Nets [Bellare and Scott 93]
- Abstract 'constant' interaction [Milner 81]
- Strategies on Games [HOBS]

Complicated
...

Game Semantics

Fully Abstract

The realm of Name Passing Interactions



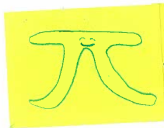
$M \approx N$



$[M] \approx [N]$



System



Session



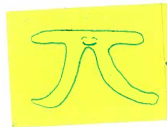
$M \approx N$



$\llbracket M \rrbracket \approx \llbracket N \rrbracket$



System



Session



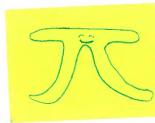
$M \approx N$



$[[M]] \approx [[N]]$



System



Session



$$M \approx N$$

$$\llbracket M \rrbracket \approx \llbracket N \rrbracket$$

$$\llbracket P \rrbracket \approx \llbracket Q \rrbracket$$



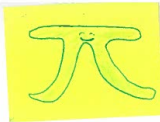
$$P \approx Q$$

Reverse

New



System
F



Session



$$M \approx N$$

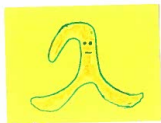
$$\llbracket M \rrbracket \approx \llbracket N \rrbracket$$

$$\llbracket P \rrbracket \approx \llbracket Q \rrbracket$$

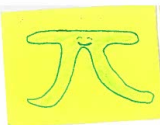


$$P \approx Q$$

Reverse
New



System
 \models



Session



$$M \approx N$$

$$\llbracket M \rrbracket \approx \llbracket N \rrbracket$$

$$\llbracket P \rrbracket \approx \llbracket Q \rrbracket$$



$$P \approx Q$$

TWO APPLICATIONS

Session π Caires, Pérez, Pfenning, Toninho 13

Types

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

Processes

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

Judgement

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

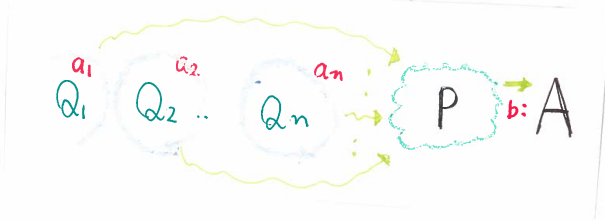
Process

Poly Vars

name Type

name type

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



Judgement

$$\underbrace{X_1, \dots, X_k}_{\text{Poly Vars}} ; \underbrace{a_1: A_1, \dots, a_n: A_n}_{\substack{\uparrow \\ \text{name} \quad \text{Type}}} \vdash \underbrace{P}_{\text{Process}} :: \underbrace{b: A}_{\substack{\uparrow \\ \text{name} \quad \text{type}}}$$

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



Judgement

$$X_1, \dots, X_k ; a_1 : A_1, \dots, a_m : A_m \vdash P :: b : A$$

Diagram annotations:
- X_1, \dots, X_k is labeled "Poly Vars".
- $a_1 : A_1, \dots, a_m : A_m$ is labeled "name" (pointing to a_i) and "Type" (pointing to A_i).
- P is labeled "Process".
- $b : A$ is labeled "name" (pointing to b) and "type" (pointing to A).

Cut Elimination

$$\frac{\Delta_1 \vdash P_1 :: a : A \quad \Delta_2, a : A \vdash P_2 :: b : B}{\Delta_1, \Delta_2 \vdash (va) (P_1 | P_2) :: b : B}$$

Identity

$$a : A \vdash [a \leftrightarrow b] :: b : A$$

Polymorphic Session

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

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

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

$\approx \pi$

Barbed
Congruence

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

Linear F

zhao, Zhang, Zdancewicz 2010



Types

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

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

Encoding is FUN 

Encoding: from  to  Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

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

$$\boxed{[M]_a}$$

\uparrow
name

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

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

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

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

The π -calculus as a Descriptive Tool

λ in π

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

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

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

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

Milner's
Encoding
1991

Encoding: from λ to π Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

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

$\llbracket M \rrbracket_a$
 \uparrow
 name

$$\llbracket x \rrbracket_a = \llbracket x \leftrightarrow a \rrbracket$$

$$\llbracket \langle \rangle \rrbracket_a = 0$$

$$\llbracket \lambda x. M \rrbracket_a = a(x). \llbracket M \rrbracket_a$$

$$\llbracket MN \rrbracket_a = \llbracket M \rrbracket_x \mid \bar{x}(y). \llbracket N \rrbracket_y \mid \llbracket x \leftrightarrow a \rrbracket$$

λ in π

$$\llbracket x \rrbracket_u \stackrel{\text{def}}{=} \bar{x}(u).$$

$$\llbracket \lambda x. M \rrbracket_u \stackrel{\text{def}}{=} u(x.u). \llbracket M \rrbracket_u.$$

$$\llbracket MN \rrbracket_u \stackrel{\text{def}}{=} (\nu \bar{z}) (\llbracket M \rrbracket_z \mid \bar{z}(u) \mid \llbracket x \leftrightarrow N \rrbracket)$$

with $\llbracket x \leftrightarrow N \rrbracket \stackrel{\text{def}}{=} !x(u). \llbracket N \rrbracket_u.$

From  to 

From Sequent Calculus to Natural Deduction

$\llbracket P \rrbracket \Delta \vdash a:A$ with $\Delta \vdash P :: a:A$

$\llbracket 0 \rrbracket = \langle \rangle$

$\llbracket [x \leftrightarrow a] \rrbracket = x$

$\llbracket a(x). P \rrbracket = \lambda x. \llbracket P \rrbracket$

$\llbracket a(x). P \rrbracket = \Lambda X. \llbracket P \rrbracket$

Parallel

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

Substitute P into a in Q

$$= \frac{\Delta_2, \alpha:A \vdash [Q]_b :: C \quad \Delta_1 \vdash [P]_a :: A}{\Delta_1, \Delta_2 \vdash [Q] \{ [P] / a \} :: C}$$

Poly

$$\left[x \langle B \rangle. P \right] = \left[P \right] \{ x[B] / x \}$$

↙ Type

Application of B to x

↑
replace x by x[B]

$$\text{cf. } \left[x \langle b \rangle. (P \mid Q) \right] = \left[Q \right] \{ (x \langle P \rangle) / x \}$$

Application of P to x

Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\llbracket M \rrbracket_z) \cong M$$

$$\llbracket (P) \rrbracket_z \cong P$$

Full Abstraction

$$\llbracket M \rrbracket_z \cong \llbracket N \rrbracket_z \text{ iff } M \cong N$$

$$(P) \cong (Q) \text{ iff } P \cong Q$$

Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\llbracket M \rrbracket_z) \cong M$$

$$\llbracket (P) \rrbracket_z \cong P$$

Definability

$$\forall P \exists M \llbracket M \rrbracket_z \cong P$$

$$\forall M \exists P (P) \cong M$$

$$\llbracket M \rrbracket_z \cong P$$

$$(P) \cong M$$

Full Abstraction

$$\llbracket M \rrbracket_z \cong \llbracket N \rrbracket_z \text{ iff } M \cong N$$

$$(P) \cong (Q) \text{ iff } P \cong Q$$

Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\llbracket M \rrbracket_z) \cong M$$



$$\llbracket (P) \rrbracket_z \cong P$$



Full Abstraction

$$\llbracket M \rrbracket_z \cong \llbracket N \rrbracket_z \text{ iff } M \cong N$$



$$(P) \cong (Q) \text{ iff } P \cong Q$$

Derived

Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\llbracket M \rrbracket_z) \cong M$$



$$\llbracket (P) \rrbracket_z \cong P$$



Full Abstraction

$$\llbracket M \rrbracket_z \cong \llbracket N \rrbracket_z$$



$$M \cong N$$

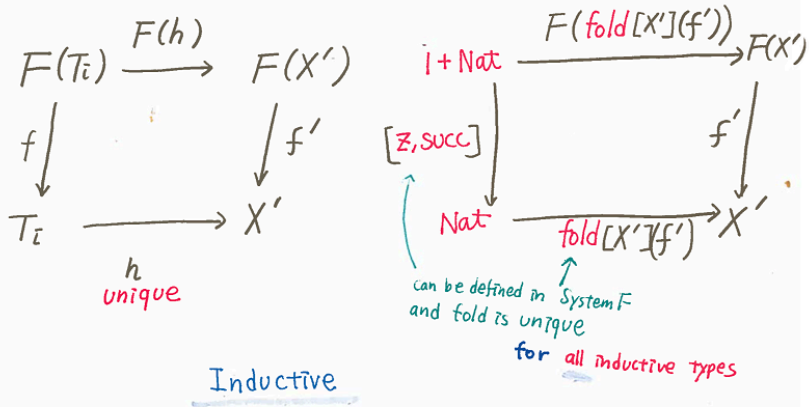
$$(P) \cong (Q)$$



$$P \cong Q$$

Application 1 Inductive and Coinductive Types

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



Coinductive Types

$$\begin{array}{ccc} X' & \xrightarrow{h} & T_f \\ f' \downarrow & & \downarrow f \\ F(X') & \xrightarrow{F(h)} & F(T_f) \end{array} \qquad \begin{array}{ccc} X' & \xrightarrow{\text{unfold } [X](f')} & \text{NatStream} \\ f' \downarrow & & \downarrow [\text{hd}, \text{tl}] \\ F(X') & \xrightarrow{F(\text{unfold } [X](f'))} & \text{Nat} \times \text{NatStream} \end{array}$$

Question Sess Poly π is Expressive Enough?

Theorems

$$\forall Q \text{ s.t. } u: F(X) \rightarrow X, y_1: T_z \vdash Q :: y_2: X. \text{Fold}(X) \cong Q$$

$$\forall_Q \text{ s.t. } u: A \rightarrow F(A), y_1: A \vdash Q :: y_2: T, Q \cong \text{Unfold}(A)$$

Application (2) HO π + Process Monad

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

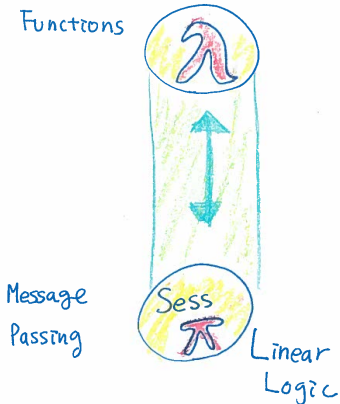
HO π

λ

$$P \rightarrow Q \quad \Rightarrow \quad (P) \dashv\dashv (Q)$$

cf. [CPPT'13] logical relation

Summary



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

Summary

SAD?



Functions



Linear
Logic

Message
Passing

LL-based

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

Summary

SAD?



Functions



Linear Logic

Message Passing

NO!



LL-based

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