on Polymorphic Sessions and Functions

a Tale of Two Encodings

Bernardo Toninho @ Nova

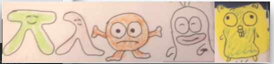
Nobuko Yoshida @ Imperial

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

Mobility Research Group

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

www.scribble.org

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.

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

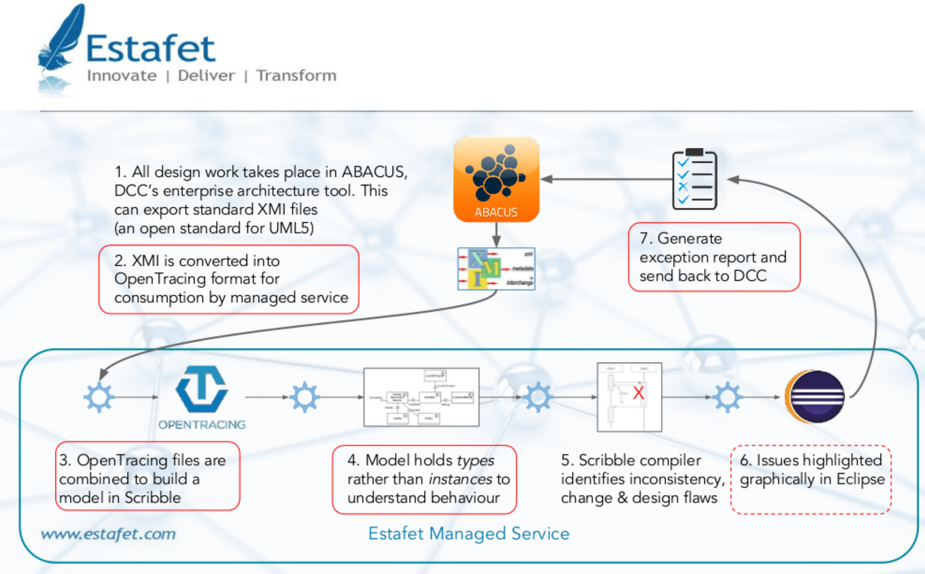
```
module examples;

global protocol HelloWorld(role Me, role World) {
  hello() from Me to World;
  choice at World {
    goodMorning1() from World to Me;
  } or {
    goodMorning1() from World to Me;
  }
}
```

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

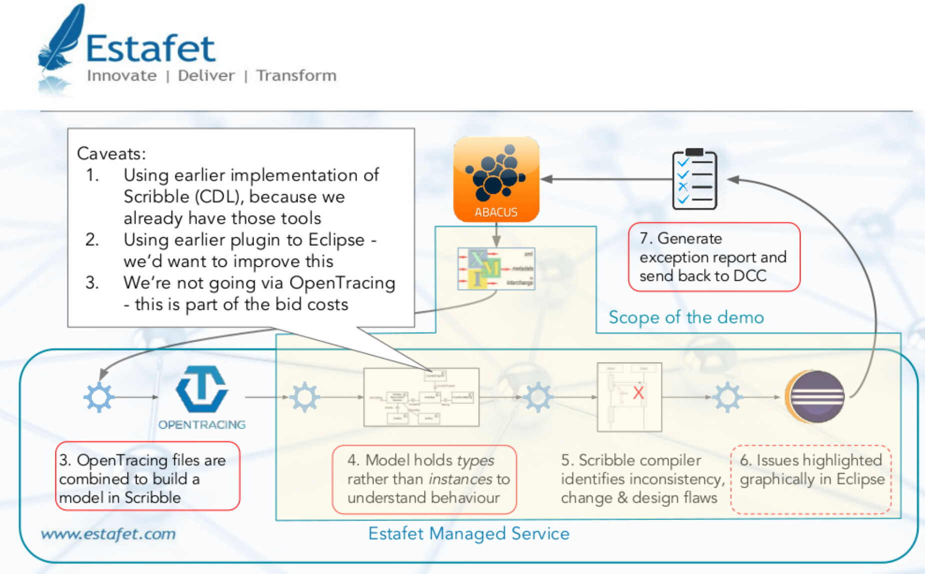
OOI Collaboration

- **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



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 low. A well-typed endpoint program using our library is ranteed to perform only compliant session I/O actions 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

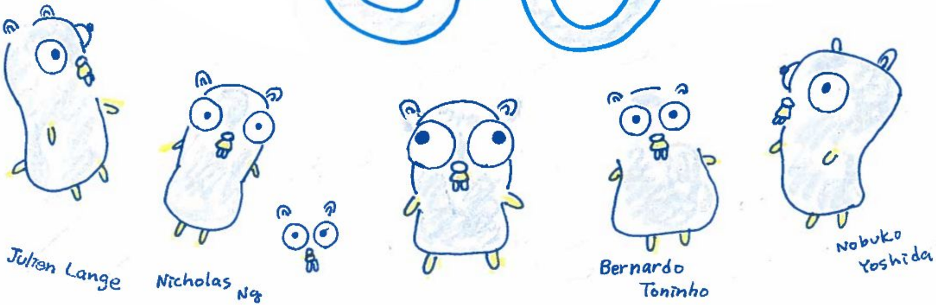
This is amazing! I guess I need to switch

weekly
on sergeytihon.com

FsharpX
F# eXchange

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

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 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

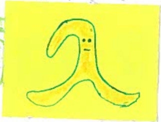
- **[LICS'18]** Romain Demangeon, NY: Casual Computational Complexity of Distributed Processes.
- **[CC'18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS'18]** Bernardo 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.

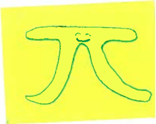
on Polymorphic Sessions and Functions

a Tale of Two Encodings

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 Functi $\lambda \to \pi$

a Tale of Two Fully Abstract Encodings

Bernardo Toninho @ Nova
Nobuko Yoshida @ Imperial

on Polymorphic Sessions and Functi[ons]

a Tale of Two Fully Abstract Encodings

$\lambda \leftarrow \pi$

New

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

The π-calculus as a Descriptive Tool

by Kohei Honda 1995

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

π $\quad P ::= \Sigma \pi_i.P_i \mid P|Q \mid (\nu x)P \mid !P \mid 0.$

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

Milner's Encoding 1991

λ in π

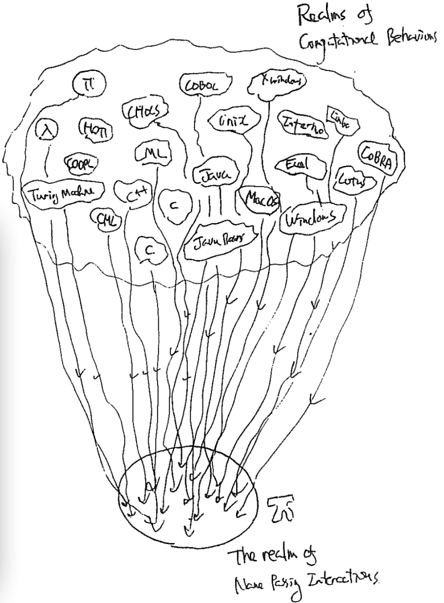
$$[\![x]\!]_u \stackrel{\text{def}}{=} \overline{x}\langle u \rangle.$$

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

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

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

* Examples of Representable Computation.



- λ-calculus [MPW89, Milner90, Milner92, …]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, …]
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [HY94]
- strategies on Games [HO95]

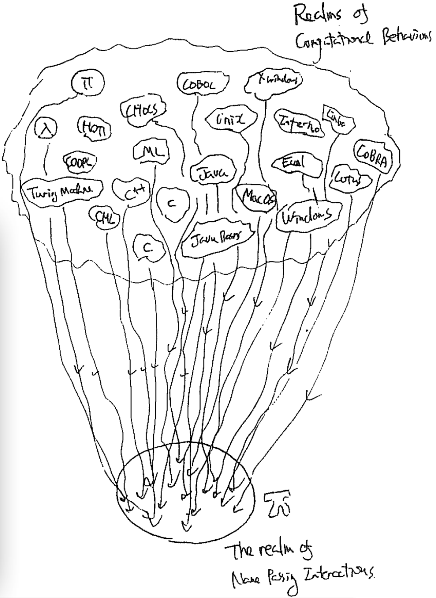
⋮

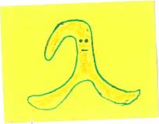
* Examples of Representable Computation.

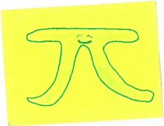
Operationally Sound but NOT Fully Abstract

- λ-calculus [MPW89, Milner90, Milner92, …]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, …]
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [HY94]
- Strategies on Games [HO95]

⋮



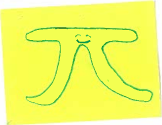




$$M \simeq N \quad \Leftarrow \quad [\![M]\!] \simeq [\![N]\!]$$

Conextual Congruence

Contextual Congruence



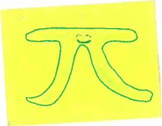
$$M \simeq N \quad \longleftarrow \quad [\![M]\!] \simeq [\![N]\!]$$

Conextual congruence

Contextual Congruence

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

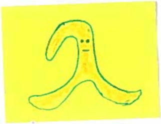
$$C[[\![M]\!]]\Downarrow_a \text{ iff } C[[\![N]\!]]\Downarrow_a$$

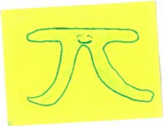


$$M \simeq N \quad \underset{\text{NOT}}{\not\Rightarrow} \quad [\![M]\!] \simeq [\![N]\!]$$

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

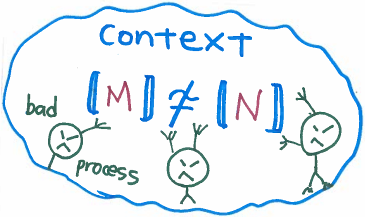
$$C[[\![M]\!]]\Downarrow_a \ \text{ iff } \ C[[\![N]\!]]\Downarrow_a$$





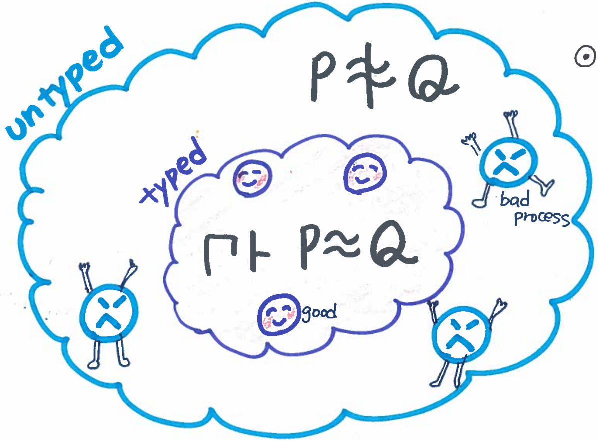
$$M \simeq N \;\not\Rightarrow\; [\![M]\!] \simeq [\![N]\!]$$

NOT



Typed Semantics in π 1991 →

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



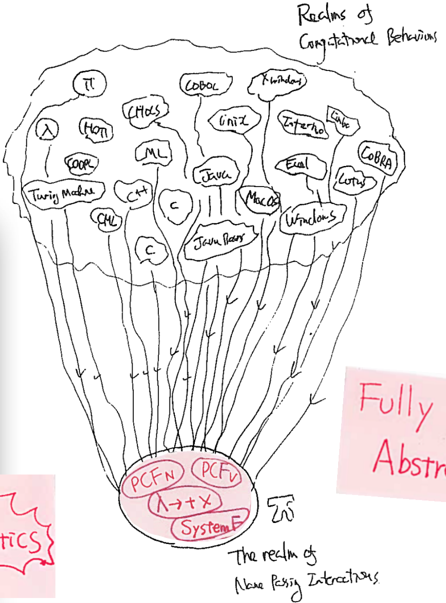
- Correctness of Encoding
- Limit environment Γ
  ⟹ Equate more processes
- Compositional

* Examples of Representable Computation.

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [HY94]
- strategies on Games [HO95]

⋮

Game Semantics



Fully Abstract

* Examples of Representable Computation.

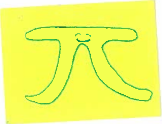
- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner S
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [
- strategies on Games [HO95]

⋮

Complicated ...

Game Semantics

Fully Abstract



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

λ System F

π Session ILL

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

λ System F

π Session ILL

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

λ System F

π Session ILL

$$M \simeq N \qquad \llbracket M \rrbracket \simeq \llbracket N \rrbracket$$

$$\llbracket P \rrbracket \simeq \llbracket Q \rrbracket \Rightarrow P \simeq Q$$

Reverse

New

λ System F

π Session ILL

$M \simeq N$

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

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

Reverse

New

λ System F

π Session ILL

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

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

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

Process

$$\underbrace{X_1, \ldots, X_k}_{\text{Poly Vars}} \; ; \; a_1 : A_1, \ldots, a_n : A_n \vdash P :: b : A$$

name Type name type

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

Judgement

Process

$$\underbrace{X_1, \ldots, X_k}_{\text{Poly Vars}} ; a_1 : A_1, \ldots, a_n : A_n \vdash P :: b : A$$

name Type name type

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

Judgement

Process

$$\underbrace{X_1, \ldots, X_k}_{\text{Poly Vars}} \; ; \; a_1 : A_1, \ldots, a_n : A_n \vdash P :: b : A$$

name Type name type

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

$\approx_\pi$

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

output

Barbed Congruence

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

Linear F

zhao, Zhang, Zdancewic 2010 

Types

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

Terms

$$M, N ::= \lambda x. M \mid M N \mid \langle M \otimes N \rangle \mid \text{let } x \otimes y = M \text{ in } N$$
$$\mid\ !M \mid \text{let } !u = M \text{ in } N$$
$$\mid \Lambda X. M \mid M[A] \mid \text{pack } A \text{ with } M \mid \text{let } (X, y) = M \text{ in } N$$
$$\mid \text{let } 1 = M \text{ in } N \mid \langle\rangle \mid \mathsf{T} \mid \mathsf{F}$$

Encoding: from λ to π

Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

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

Encoding is FUN

Encoding: from λ to π

Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

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

$$\boxed{[\![M]\!]_{\underset{\text{name}}{\uparrow}{a}}}$$

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

$$[\![\langle\rangle]\!]_a = 0 \qquad\qquad [\![\lambda x. M]\!]_a = a(x).\, [\![M]\!]_a$$

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

The π-calculus as a Descriptive Tool

λ in π

Milner's Encoding 1991

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

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

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

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

Encoding: from λ to π

Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

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

$$\llbracket\ M\ \rrbracket_a$$

a ↑ name

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

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

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

$$\llbracket MN \rrbracket_a = \llbracket M \rrbracket_x \mid \overline{x}(y).\,\llbracket N \rrbracket_y \mid [x \leftrightarrow a]$$

λ in π

$$\llbracket x \rrbracket_u \stackrel{\text{def}}{=} \overline{x}\langle u\rangle.$$

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

$$\llbracket MN \rrbracket_u \stackrel{\text{def}}{=} (\nu f x)\left(\llbracket M \rrbracket_f \mid \overline{f}\langle x u\rangle \mid \llbracket x{=}N \rrbracket\right)$$

with $\llbracket x{=}N \rrbracket \stackrel{\text{def}}{=} !x(u').\,\llbracket N \rrbracket_{u'}.$

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

$$[\![\, \underbrace{x\langle B\rangle}_{\text{Application of } B \text{ to } x} . P \,]\!] = [\![P]\!]\{x[B]/x\}$$

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

Inverse

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

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

Definability

$$\forall P \; \exists M \quad [\![M]\!]_z \cong P$$

$$\forall M \; \exists P \quad (\!| P |\!) \cong M$$

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

$$\llparenthesis \llbracket M \rrbracket_Z \rrparenthesis \cong M \quad ✓$$

$$\llbracket \llparenthesis P \rrparenthesis \rrbracket_Z \cong P \quad ✓$$

Full Abstraction

$$\llbracket M \rrbracket_Z \cong \llbracket N \rrbracket_Z \Rightarrow M \cong N$$

$$\llparenthesis P \rrparenthesis \cong \llparenthesis Q \rrparenthesis \Rightarrow P \cong Q$$

Application 1 Inductive and Coinductive Types

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

$$\begin{tikzcd}
F(T_i) \arrow[r, "F(h)"] \arrow[d, "f"'] & F(X') \arrow[d, "f'"] \\
T_i \arrow[r, "h"'] & X'
\end{tikzcd}$$

h unique

$$\begin{tikzcd}
1+\mathrm{Nat} \arrow[r, "F(\mathrm{fold}[X'](f'))"] \arrow[d, "{[z,\mathrm{succ}]}"'] & F(X') \arrow[d, "f'"] \\
\mathrm{Nat} \arrow[r, "\mathrm{fold}[X'](f')"'] & X'
\end{tikzcd}$$

can be defined in System F and fold is unique

for all inductive types

Inductive

Coinductive Types

$$\begin{tikzcd} X' \arrow[r, "h"] \arrow[d, "f'"'] & T_f \arrow[d, "f"] \\ F(X') \arrow[r, "F(h)"'] & F(T_f) \end{tikzcd} \qquad \begin{tikzcd} X' \arrow[r, "\mathrm{unfold}\,[X](f')"] \arrow[d, "f'"] & \mathrm{NatStream} \arrow[d, "{[hd, tl]}"'] \\ F(X') \arrow[r, "F(\mathrm{unfold}\,[X](f'))"'] & \mathrm{Nat} \times \mathrm{NatStream} \end{tikzcd}$$

Question Sess Poly π is Expressive Enough?

Theorems

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

$\forall_A Q$ s.t. $u : A \multimap F(A),\ y_1 : A \vdash Q :: y_2 : T,$ $Q \cong \mathrm{Unfold}(A)$

Application (2) HOπ + PRosess Monad

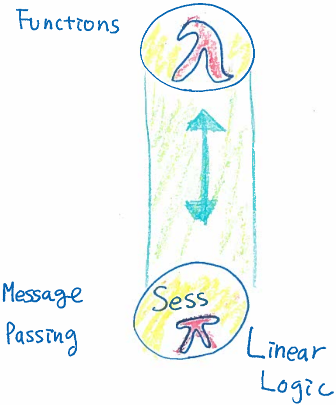
- Full Abstraction / Inverse $\quad x\langle M\rangle.\,P$
- Strong Normalisation via Strong Normalisation

HOπ $\qquad\qquad\qquad\qquad\qquad$ λ

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

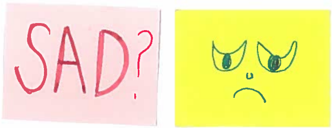
cf. [CPPT'13] logical relation

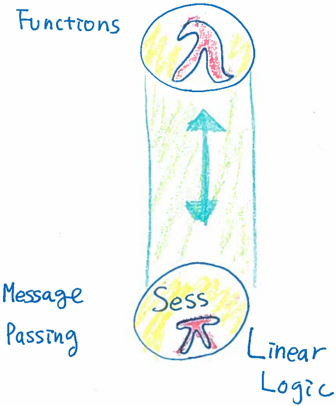
Summary



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

Summary





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

Summary

SAD?

NO!

Functions

Message Passing

Linear Logic

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