Mechanising Session Types Onwards and Upwards

Francisco Ferreira and Lorenzo Gheri
(joint work with David Castro, and Nobuko Yoshida)

2019
ABCD Meeting

Imperial College London

The First Step

- Do a case study:
- Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited, by Yoshida and Vasconcelos.

The send receive system and its cousin the relaxed and the revisited system.

Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 171 (2007) 73–93
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: *Two Systems for Higher-Order Session Communication*

Nobuko Yoshida[1]
Imperial College London

Vasco T. Vasconcelos[2]
University of Lisbon

Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 171 (2007) 73–93
www.elsevier.com/locate/entcs

Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: *Two Systems for Higher-Order Session Communica*

Nobuko Yoshida[1]
Imperial College London
Vasco T. Vasconcelos[2]
University of Lisbon

This is the first step.
Spoiler: Multiparty session types are next.

What do we have?

- A proof of type preservation formalised in Coq using `ssreflect`.
- A library to implement locally nameless with multiple name scopes and handle environments in a versatile way.
- We have a TACAS 2020 submission describing our tool.
- We built some in-team expertise (i.e. we learned some hard lessons while struggling to finish the proof).

What did we mechanise?

A tale of three systems

- We set out to represent the three systems described in the paper:
  - The Honda, Vasconcelos, Kubo system from ESOP'98

A tale of three systems

- We set out to represent the three systems described in the paper:
  - The Honda, Vasconcelos, Kubo system from ESOP'98
  - Its naïve but ultimately unsound extension

A tale of three systems

- We set out to represent the three systems described in the paper:
  - The Honda, Vasconcelos, Kubo system from ESOP'98
  - Its naïve but ultimately unsound extension
  - Its revised system inspired by Gay and Hole in Acta Informatica

The Send Receive System

$$
\begin{array}{rll}
P & ::= & \texttt{request}\ a(k)\ \texttt{in}\ P & \text{session request} \\
& | & \texttt{accept}\ a(k)\ \texttt{in}\ P & \text{session acceptance} \\
& | & k![\tilde{e}]; P & \text{data sending} \\
& | & k?(\tilde{x})\ \texttt{in}\ P & \text{data reception} \\
& | & k \lhd l; P & \text{label selection} \\
& | & k \rhd \{l_1 : P_1 [\!] \cdots [\!] l_n : P_n\} & \text{label branching} \\
& | & \texttt{throw}\ k[k']; P & \text{channel sending} \\
& | & \texttt{catch}\ k(k')\ \texttt{in}\ P & \text{channel reception} \\
& | & \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q & \text{conditional branch} \\
& | & P \mid Q & \text{parallel composition} \\
& | & \texttt{inact} & \text{inaction} \\
& | & (\nu u)P & \text{name/channel hiding} \\
& | & \texttt{def}\ D\ \texttt{in}\ P & \text{recursion} \\
& | & X[\tilde{e}\tilde{k}] & \text{process variables} \\
e & ::= & c & \text{constant} \\
& | & e + e' \ \mid\ e - e' \ \mid\ e \times e \ \mid\ \texttt{not}(e) \ \mid\ \ldots & \text{operators} \\
D & ::= & X_1(\tilde{x}_1\tilde{k}_1) = P_1\ \texttt{and} \cdots \texttt{and}\ X_n(\tilde{x}_n\tilde{k}_n) = P_n & \text{declaration for recursion}
\end{array}
$$

The Send Receive System

$$
\begin{array}{rcll}
P & ::= & \texttt{request}\ a(k)\ \texttt{in}\ P & \text{session request} \\
& | & \texttt{accept}\ a(k)\ \texttt{in}\ P & \text{session acceptance} \\
& | & k![\tilde{e}]; P & \text{data sending} \\
& | & k?(\tilde{x})\ \texttt{in}\ P & \text{data reception} \\
& | & k \lhd l; P & \text{label selection} \\
& | & k \rhd \{l_1 : P_1 [\!] \cdots [\!] l_n : P_n\} & \text{label branching} \\
& | & \texttt{throw}\ k[k']; P & \text{channel sending} \\
& | & \texttt{catch}\ k(k')\ \texttt{in}\ P & \text{channel reception} \\
& | & \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q & \text{conditional branch} \\
& | & P \mid Q & \text{parallel composition} \\
& | & \texttt{inact} & \text{inaction} \\
& | & (\nu u)P & \text{name/channel hiding} \\
& | & \texttt{def}\ D\ \texttt{in}\ P & \text{recursion} \\
& | & X[\tilde{e}\tilde{k}] & \text{process variables} \\
e & ::= & c & \text{constant} \\
& | & e + e' \ \mid\ e - e' \ \mid\ e \times e \ \mid\ \texttt{not}(e) \ \mid\ \ldots & \text{operators} \\
D & ::= & X_1(\tilde{x}_1\tilde{k}_1) = P_1\ \texttt{and} \cdots \texttt{and}\ X_n(\tilde{x}_n\tilde{k}_n) = P_n & \text{declaration for recursion}
\end{array}
$$

The Send Receive System

We consider terms up-to α-conversion

$$
\begin{array}{rcll}
 & & \texttt{est}\ a(k)\ \texttt{in}\ P & \text{session request} \\
 & & \texttt{t}\ a(k)\ \texttt{in}\ P & \text{session acceptance} \\
 & & & \text{data sending} \\
 & & \texttt{in}\ P & \text{data reception} \\
 & &]; P & \text{label selection} \\
 & & \mid k \rhd \{l_1 : P_1 \,[\!]\, \cdots \,[\!]\, l_n : P_n\} & \text{label branching} \\
 & & \mid \texttt{throw}\ k[k']; P & \text{channel sending} \\
 & & \mid \texttt{catch}\ k(k')\ \texttt{in}\ P & \text{channel reception} \\
 & & \mid \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q & \text{conditional branch} \\
 & & \mid P \mid Q & \text{parallel composition} \\
 & & \mid \texttt{inact} & \text{inaction} \\
 & & \mid (\nu u)P & \text{name/channel hiding} \\
 & & \mid \texttt{def}\ D\ \texttt{in}\ P & \text{recursion} \\
 & & \mid X[\tilde{e}\tilde{k}] & \text{process variables} \\
e & ::= & c & \text{constant} \\
 & & \mid e + e' \ \mid\ e - e' \ \mid\ e \times e \ \mid\ \texttt{not}(e) \ \mid\ \ldots & \text{operators} \\
D & ::= & X_1(\tilde{x}_1\tilde{k}_1) = P_1\ \texttt{and} \cdots \texttt{and}\ X_n(\tilde{x}_n\tilde{k}_n) = P_n & \text{declaration for recursion}
\end{array}
$$

The Send Receive System

We consider terms up-to α-conversion

$$
\begin{array}{rcll}
 & & \texttt{uest}\ a(k)\ \texttt{in}\ P & \text{session request} \\
 & & \texttt{t}\ a(k)\ \texttt{in}\ P & \text{session acceptance} \\
 & & \texttt{in}\ P & \text{data sending} \\
 & &];P & \text{data reception} \\
 & & k \triangleright \{l_1 : P_1 \,[\!]\, \cdots \,[\!]\, l_n : P_n\} & \text{label selection} \\
 & & \texttt{throw}\ k[k'];P & \text{label branching} \\
 & & \texttt{catch}\ k(k')\ \texttt{in}\ P & \\
 & & \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q & \\
 & & P \mid Q & \\
 & & \texttt{inact} & \\
 & & (\nu u)P & \\
 & & \texttt{def}\ D\ \texttt{in}\ P & \\
 & & X[\tilde{e}\tilde{k}] & \\
e & ::= & c & \\
 & & e + e' \mid e - e' \mid e \times e \mid \texttt{not}(e) \mid \ldots & \\
D & ::= & X_1(\tilde{x}_1\tilde{k}_1) = P_1\ \texttt{and} \cdots \texttt{and}\ X_n(\tilde{x}_n\tilde{k}_n) = P_n & \text{declara}
\end{array}
$$

Then we cannot distinguish:

`k?(x) in inact`

and

`k?(y) in inact`

α-conversion curse or Blessing?

$$(\texttt{throw}\ k[k'];P_1) \mid (\texttt{catch}\ k(k')\ \texttt{in}\ P_2) \rightarrow P_1 \mid P_2$$

- The original system depends crucially on names

α-conversion curse or Blessing?

$$(\texttt{throw}\ k[k'];P_1) \mid (\texttt{catch}\ k(k')\ \texttt{in}\ P_2) \rightarrow P_1 \mid P_2$$

- The original system depends crucially on names

α-conversion curse or Blessing?

$$(\texttt{throw}\ k[k'];P_1) \mid (\texttt{catch}\ k(k')\ \texttt{in}\ P_2) \rightarrow P_1 \mid P_2$$

- The original system depends crucially on names

This is a bound variable.

α-conversion curse or Blessing?

$$(\texttt{throw}\ k[k'];P_1) \mid\ (\texttt{catch}\ k(k')\ \texttt{in}\ P_2) \ \rightarrow\ P_1 \mid\ P_2$$

- The original system depends crucially on names

This is a bound variable.

- If α-conversion is built in, this rule collapses to:

$$(\texttt{throw}\ k[k'];P_1) \mid\ (\texttt{catch}\ k(k'')\ \texttt{in}\ P_2) \ \rightarrow\ P_1 \mid\ P_2[k'/k'']$$

The Naïve Representation

The Naïve Representation

- It "**looks like**" the original Send Receive system.

The Naïve Representation

- It "**looks like**" the original Send Receive system.
- You start **suspecting** is wrong when defining the reduction relation.

The Naïve Representation

- It "**looks like**" the original Send Receive system.

- You start **suspecting** is wrong when defining the reduction relation.

- You **know** there is a problem when the proof fails.

We have to discuss Adequacy

- I see this problem in one of two ways:
  - Either, we require proofs of adequacy.
  - Or we consider the meaning of the mechanisation "first-class".

We have to discuss Adequacy

JFP **17** (4 & 5): 613–673, 2007.
doi:10.1017/S0956796807006430 First published online 6 July 2007 Printed in the United Kingdom

Mechanizing metatheory in a logical framework

ROBERT HARPER and DANIEL R. LICATA
Carnegie Mellon University, Pittsburgh, PA 15213, USA
(*e-mail:* {rwh,drl}@cs.cmu.edu)

class".

We have to discuss Adequacy

18 pages dedicated to the proof for the STLC!

JFP **17** (4 & 5): 613–673, 2007. © 2007 Cambridge University Press
doi:10.1017/S0956796807006430 First published online 6 July 2007 Printed in the United Kingdom

Mechanizing metatheory in a logical framework

ROBERT HARPER and DANIEL R. LICATA
Carnegie Mellon University, Pittsburgh, PA 15213, USA
(*e-mail:* {rwh,drl}@cs.cmu.edu)

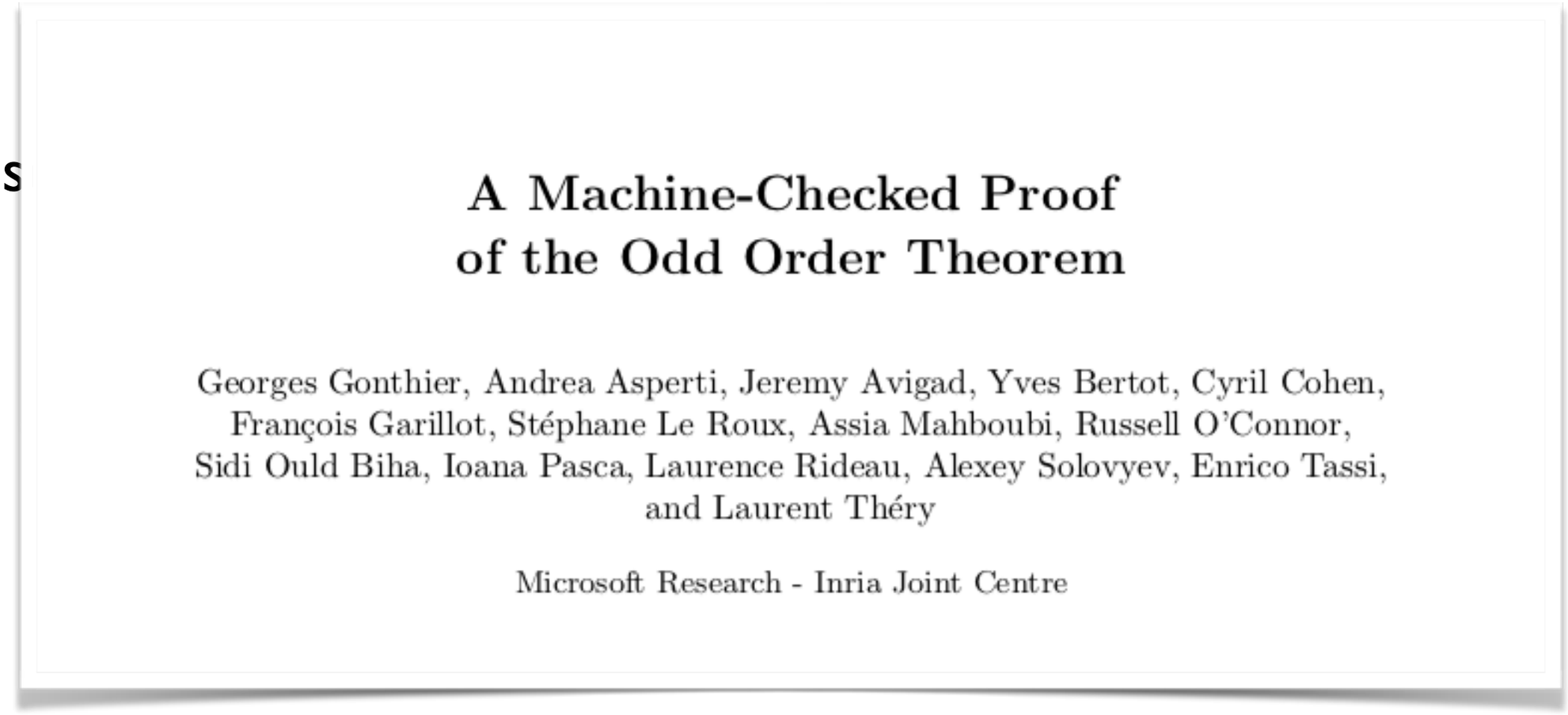
- Class".

We have to discuss Adequacy

- I s

-

-



A Machine-Checked Proof
of the Odd Order Theorem

Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen,
François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor,
Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi,
and Laurent Théry

Microsoft Research - Inria Joint Centre

We have to discuss Adequacy

- I see this problem in one of two ways:
  - Either, we require proofs of adequacy.
  - Or we consider the meaning of the mechanisation "first-class".

The Revisited system

- Now we distinguish between the endpoints of channels.

- It can be represented with LN-variables and names.

Four kinds of atoms

```
Inductive proc : Set :=
 | request : scvar → proc →  proc
 | accept : scvar → proc →  proc

 | send : channel →  exp →  proc →  proc
 | receive : channel → proc →  proc

 | select :
   channel →  label →  proc →  proc
 | branch :
   channel →  proc →  proc →  proc

 | throw :
   channel →  channel →  proc →  proc
 | catch : channel → proc →  proc

 | ife : exp →  proc →  proc →  proc
 | par : proc →  proc →  proc
 | inact : proc

 (* hides a channel name *)
 | nu_ch : proc →  proc
 (* hides a name *)
 | nu_nm : proc →  proc
 (* process replication *)
 | bang : proc →  proc
 .
```

binds variable from $\mathbb{A}_{SC}$

binds variable from $\mathbb{A}_{EV}$

binds variable from $\mathbb{A}_{LC}$

binds channel from $\mathbb{A}_{CN}$

Four kinds of atoms

```
Inductive proc : Set :=
 | request : scvar → proc →  proc
 | accept : scvar → proc →  proc

 | send : channel →  exp →  proc →  proc
 | receive : channel → proc →  proc

 | select :
   channel →  label →  proc →  proc
 | branch :
   channel →  proc →  proc →  proc

 | throw :
   channel →  channel →  proc →  proc
 | catch : channel → proc →  proc

 | ife : exp →  proc →  proc →  proc
 | par : proc →  proc →  proc
 | inact : proc

 (* hides a channel name *)
 | nu_ch : proc →  proc
 (* hides a name *)
 | nu_nm : proc →  proc
 (* process replication *)
 | bang : proc →  proc
 .
```

binds variable from $\mathbb{A}_{SC}$

binds variable from $\mathbb{A}_{EV}$

binds variable from $\mathbb{A}_{LC}$

binds channel from $\mathbb{A}_{CN}$

Four kinds of atoms

```
Inductive proc : Set :=
 | request : scvar → proc →  proc
 | accept : scvar → proc →  proc

 | send : channel →  exp →  proc →  proc
 | receive : channel → proc →  proc

 | select :
   channel →  label →  proc →  proc
 | branch :
   channel →  proc →  proc →  proc

 | throw :
   channel →  channel →  proc →  proc
 | catch : channel → proc →  proc

 | ife : exp →  proc →  proc →  proc
 | par : proc →  proc →  proc
 | inact : proc

(* hides a channel name *)
 | nu_ch : proc →  proc
(* hides a name *)
 | nu_nm : proc →  proc
(* process replication *)
 | bang : proc →  proc
.
```

binds variable from $\mathbb{A}_{SC}$

binds variable from $\mathbb{A}_{EV}$

binds variable from $\mathbb{A}_{LC}$

binds channel from $\mathbb{A}_{CN}$

Four kinds of atoms

```
Inductive proc : Set :=
 | request : scvar → proc →  proc
 | accept : scvar → proc →  proc

 | send : channel →  exp →  proc →  proc
 | receive : channel → proc →  proc

 | select :
   channel →  label →  proc →  proc
 | branch :
   channel →  proc →  proc →  proc

 | throw :
   channel →  channel →  proc →  proc
 | catch : channel → proc →  proc

 | ife : exp →  proc →  proc →  proc
 | par : proc →  proc →  proc
 | inact : proc

 (* hides a channel name *)
 | nu_ch : proc →  proc
 (* hides a name *)
 | nu_nm : proc →  proc
 (* process replication *)
 | bang : proc →  proc
 .
```

binds variable from $\mathbb{A}_{SC}$

binds variable from $\mathbb{A}_{EV}$

binds variable from $\mathbb{A}_{LC}$

binds channel from $\mathbb{A}_{CN}$

Four kinds of atoms

```
Inductive proc : Set :=
 | request : scvar → proc →  proc
 | accept : scvar → proc →  proc

 | send : channel →  exp →  proc →  proc
 | receive : channel → proc →  proc

 | select :
   channel →  label →  proc →  proc
 | branch :
   channel →  proc →  proc →  proc

 | throw :
   channel →  channel →  proc →  proc
 | catch : channel → proc →  proc

 | ife : exp →  proc →  proc →  proc
 | par : proc →  proc →  proc
 | inact : proc

 (* hides a channel name *)
 | nu_ch : proc →  proc
 (* hides a name *)
 | nu_nm : proc →  proc
 (* process replication *)
 | bang : proc →  proc
 .
```

binds variable from $\mathbb{A}_{SC}$

binds variable from $\mathbb{A}_{EV}$

binds variable from $\mathbb{A}_{LC}$

binds channel from $\mathbb{A}_{CN}$

Typing environments

- Store their assumptions in a unique order
  (easy to compare)

- Only store unique assumptions
  (easy to split)

- They come with many lemmas
  (less induction proofs)

Typing environments

- Store their assumptions in a unique order **(easy to compare)**

- Only store unique assumptions **(easy to split)**

- They come with many lemmas **(less induction proofs)**

These are generic enough and easy to use. #artefact

Subject Reduction

Theorem 3.3 (Subject Reduction) *If* $\Theta; \Gamma \vdash P \triangleright \Delta$ *with* Δ *balanced and* $P \rightarrow^* Q$*, then* $\Theta; \Gamma \vdash Q \triangleright \Delta'$ *and* Δ' *balanced.*

Is straightforward to represent:

```
Theorem SubjectReduction G P Q D:
  oft G P D → balanced D → P ⟶* Q → exists D', balanced D' /\ oft G Q D'.
```

We have a tech report and a repository for the proof.

- The code for the proof can be found at:
  - https://github.com/emtst/
- We have a technical report:
  - Engineering the Meta-Theory of Session Types
  - at: https://www.doc.ic.ac.uk/research/technicalreports/2019/DTRS19-4.pdf

Onwards and Upwards

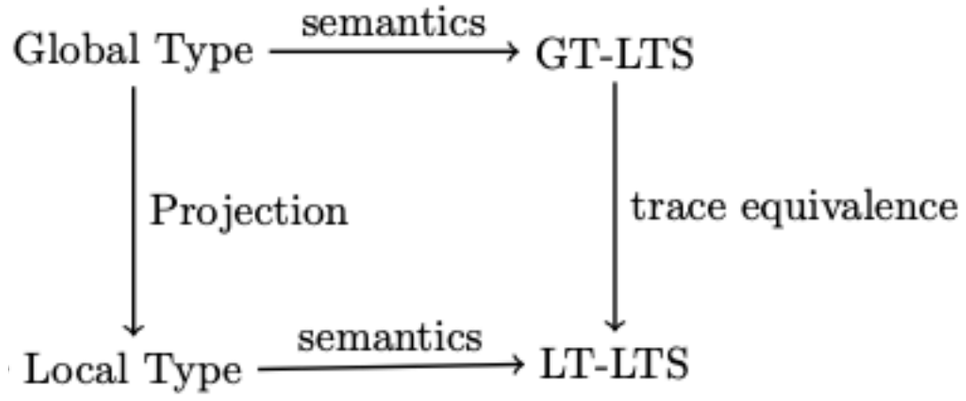
We are moving to Multiparty Session Types

- Lessons learned:
  - Doing a complete calculus just to have a similar calculus to the literature takes a lot of effort.
  - Locally nameless worked well. Particularly/Even with the multiple name scopes.
  - Mechanising proof is great, but if one squints mechanisation is akin to **very** careful implementation.

MPST

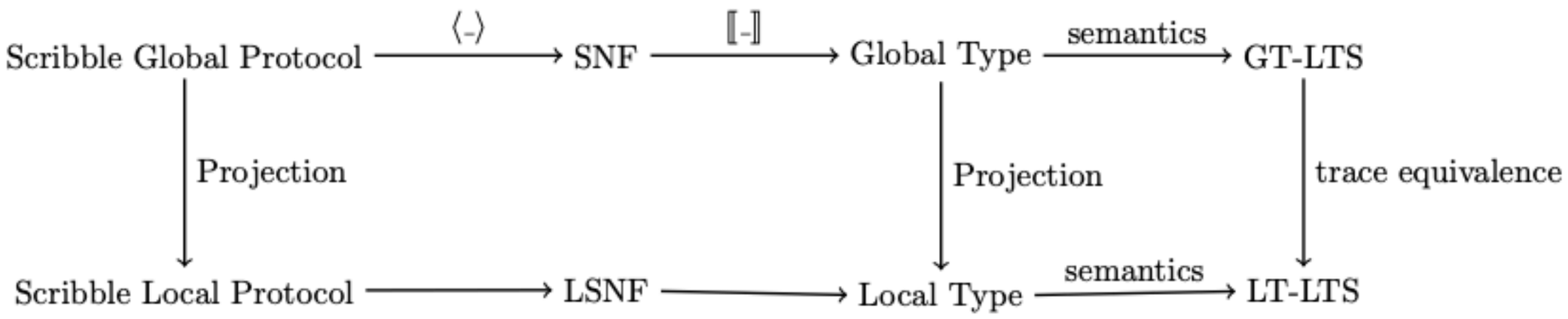
- There's four of us now: David, Francisco, Lorenzo, and Nobuko.
- We are mechanising the meta-theory of multiparty session types.
- We will build upon our locally nameless and environment implementation.
- We plan to extract certified implementations from the proofs.

Certified MPST



Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types
Deniélou, Yoshida, 2013

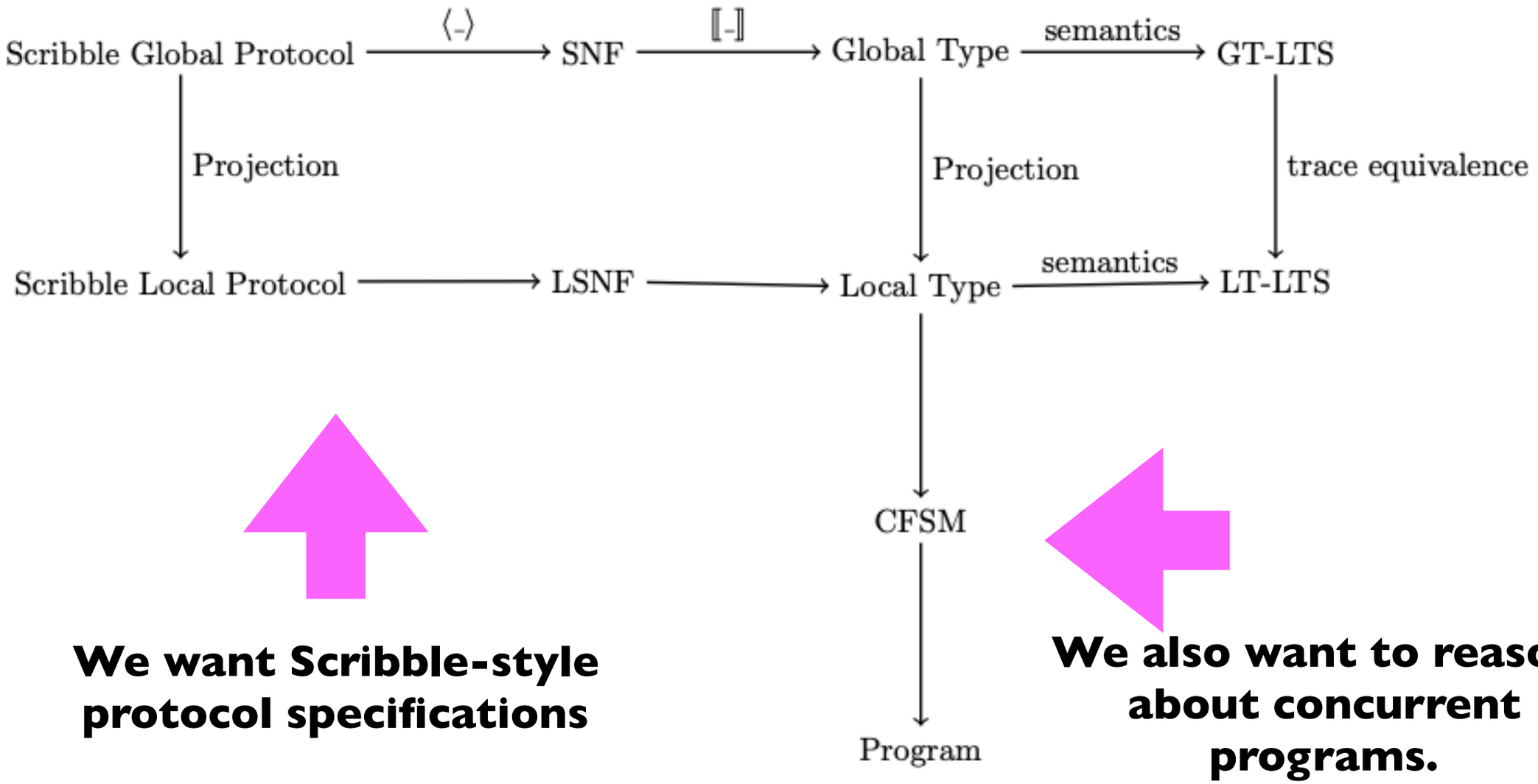
Certified MPST



We want Scribble-style protocol specifications

Featherweight Scribble, Neykova, Yoshida, 2019

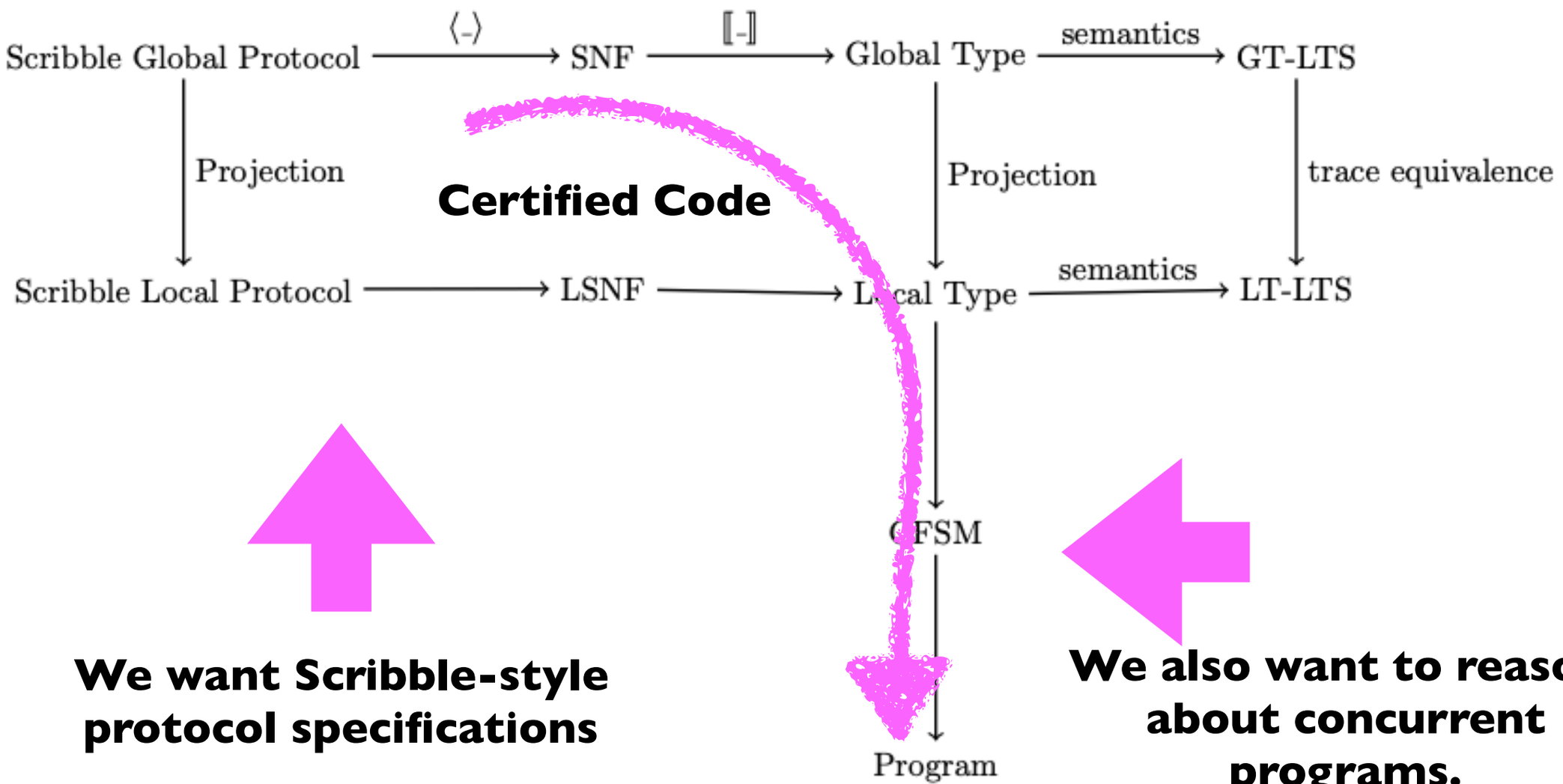
Certified MPST



We want Scribble-style protocol specifications

We also want to reason about concurrent programs.

Certified MPST



We want Scribble-style protocol specifications

We also want to reason about concurrent programs.

Mechanical Progress

- We talked about the binary session types meta-theory proof we formalised.

- We talked about our current project and our future plans.

Mechanical Progress

- We talked about the eory proof we formalised.

- We talked about our re plans.

Thanks for your kind attention! Questions?