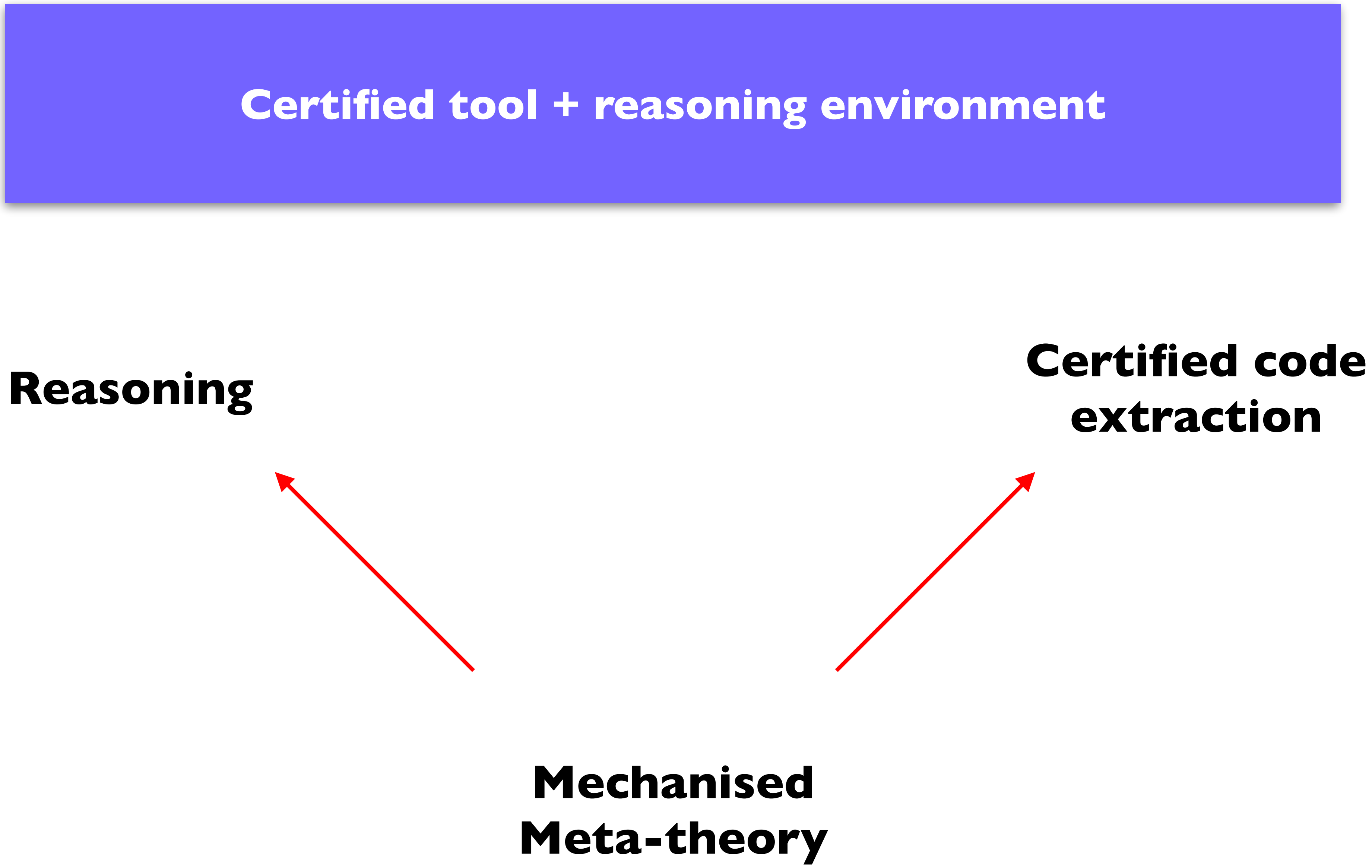
It Takes a Village: Reasoning About Concurrent Processes

David Castro, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida

2020
VEST Workshop

Imperial College London

Motivating Meta-Theory



Certified tool + reasoning environment

Reasoning

Certified code extraction

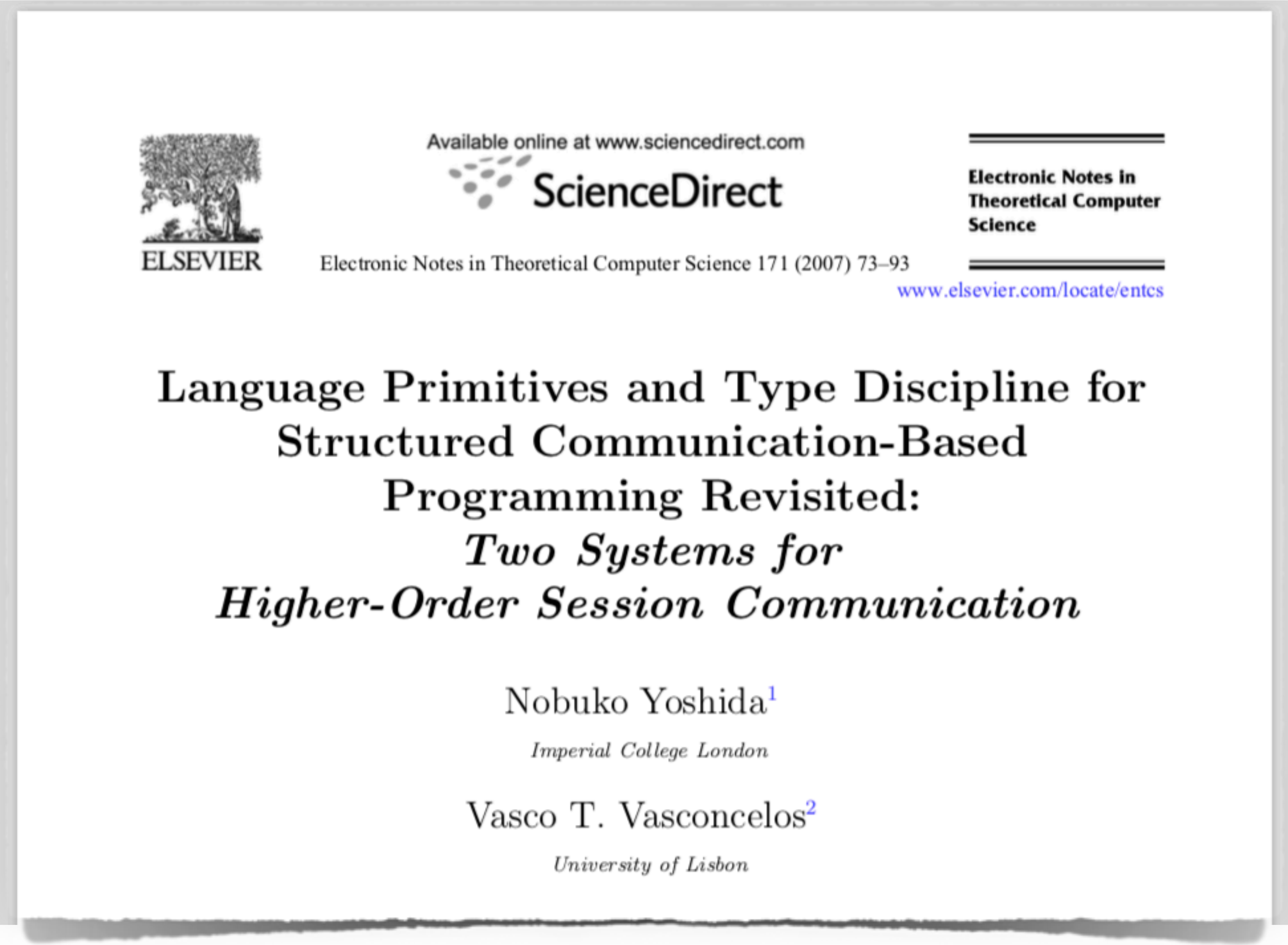
Mechanised Meta-theory

Binary Session Types

- Do a case study:
    - Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited, by Yoshida and Vasconcelos, 2007.

s n

t .



Available online at www.sciencedirect.com

ScienceDirect

ELSEVIER

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

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.

- 😃 TACAS 2020 **accepter paper and artefact** describing our tool and mechanisation.

- We built 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
  - Its naïve but ultimately unsound extension
  - Its revised system inspired by Gay and Hole in Acta Informatica

The Send Receive System

We consider terms up-to α-conversion

…est $a(k)$ in P session request

…t $a(k)$ in P session acceptance

data sending

…in P data reception

…; P label selection

| $k \triangleright \{l_1 : P_1 [\!] \cdots [\!] l_n : P_n\}$ label branching

| throw $k[k']; P$

| catch $k(k')$ in P

| if e then P else Q

| $P \mid Q$

| inact

| $(\nu u)P$

| def D in P

| $X[\tilde{e}\tilde{k}]$

e ::= c

| $e + e'$ | $e - e'$ | $e \times e$ | not(e) | …

D ::= $X_1(\tilde{x}_1\tilde{k}_1) = P_1$ and ⋯ and $X_n(\tilde{x}_n\tilde{k}_n) = P_n$ declara…

Then we cannot distinguish:
k?(x) in inact
and
k?(y) in inact

α-conversion curse or Blessing?

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

- The original system depends cruci on names

This is a bound variable.

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

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

α-conversion curse or Blessing?

- Humans have to pretend not to see the different bound names.
- However, there exist several representations that offer inherently α-convertible terms:
    - de Bruijn indices (or levels)
    - Higher Order Abstract Syntax
    - Locally Nameless

My personal take: α-conversion is more interesting that I originally gave it credit for.

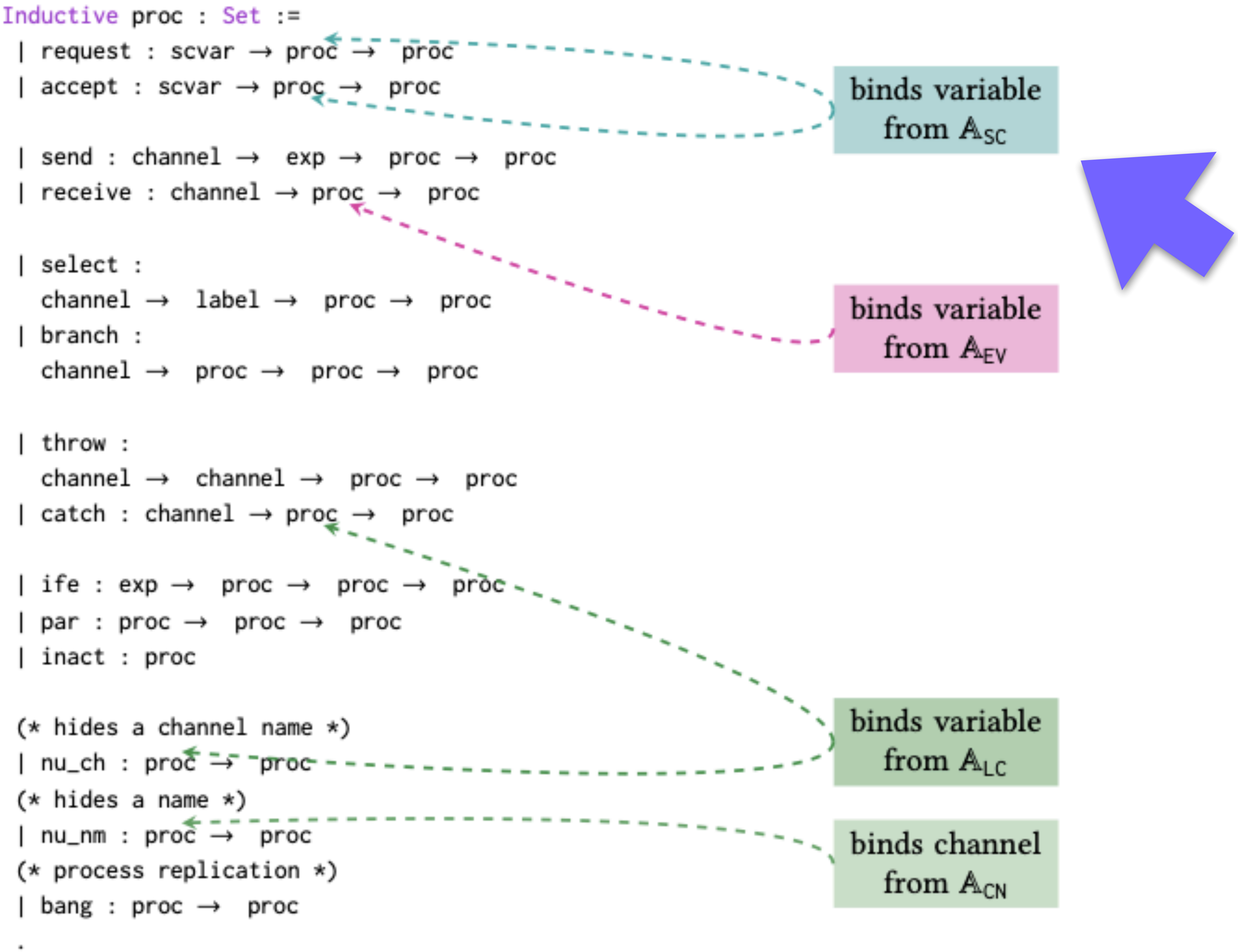
The Naïve Representation

- 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.

The Revisited system

- Now we distinguish between the endpoints of channels.
- It can be readily 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 $\mathbb{A}_{CN}$

Typing environments

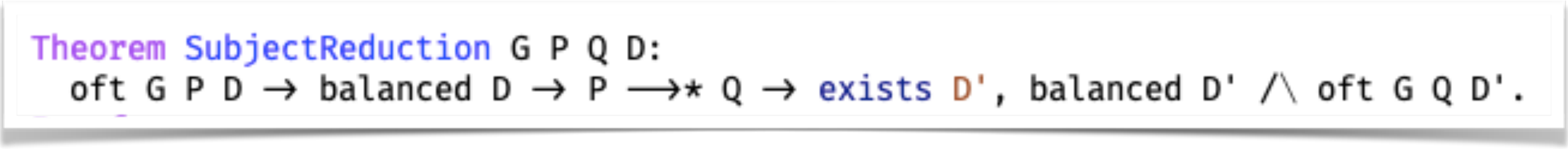
- Store their assumptions in a unique order
  (easy to compare)
- Only store unique assumptions
  (easy to split)
- They come with many lemmas
  (less induction proofs)

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* $P \to^* Q$, *then* $\Theta; \Gamma \vdash Q \triangleright \Delta'$ *and* Δ' *balanced.*

Is straightforward to represent:

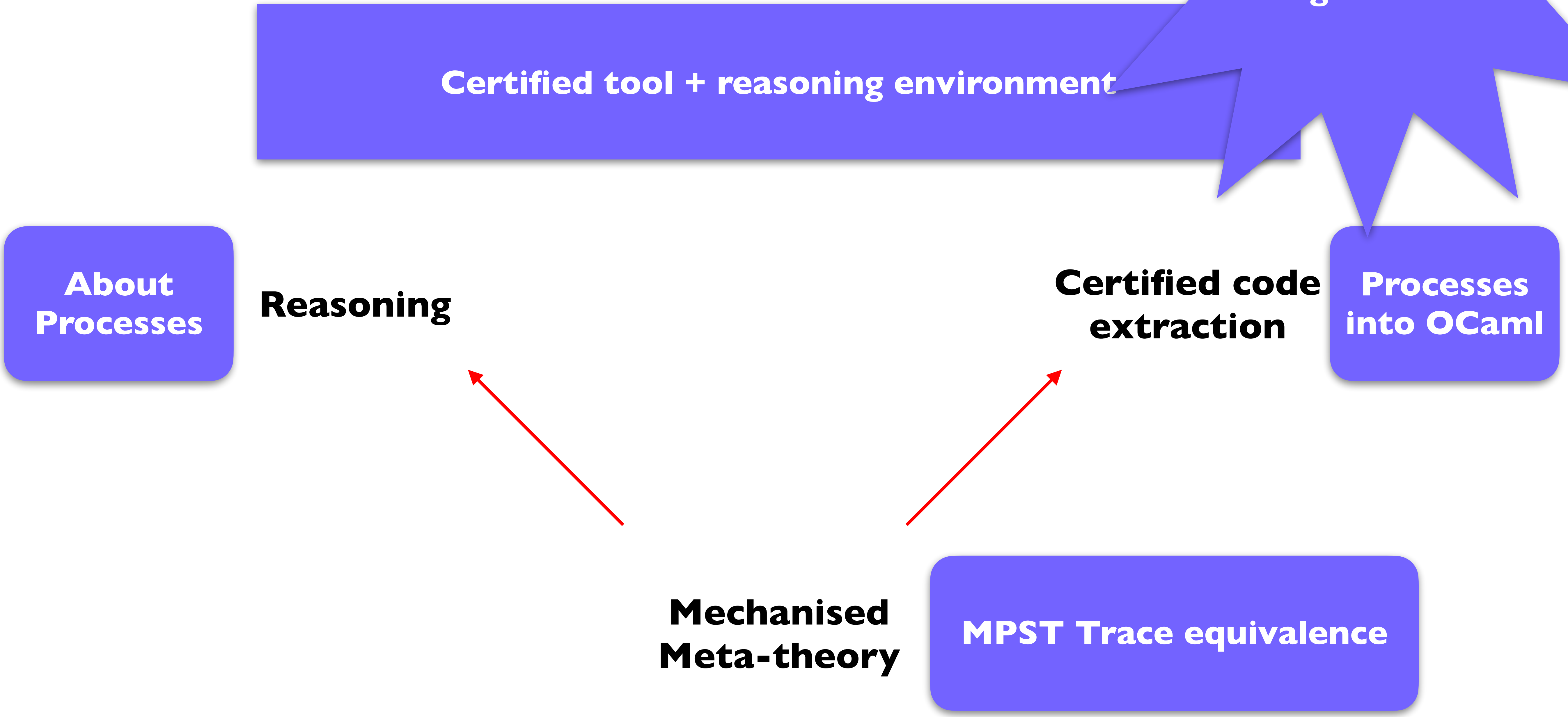


```
Theorem SubjectReduction G P Q D:
  oft G P D → balanced D → P ⟶* Q → exists D', balanced D' /\ oft G Q D'.
```

We want more from our mechanisation.

Motivating Meta-Theory

Certified Scribble Algorithms



Certified tool + reasoning environment

About Processes

Reasoning

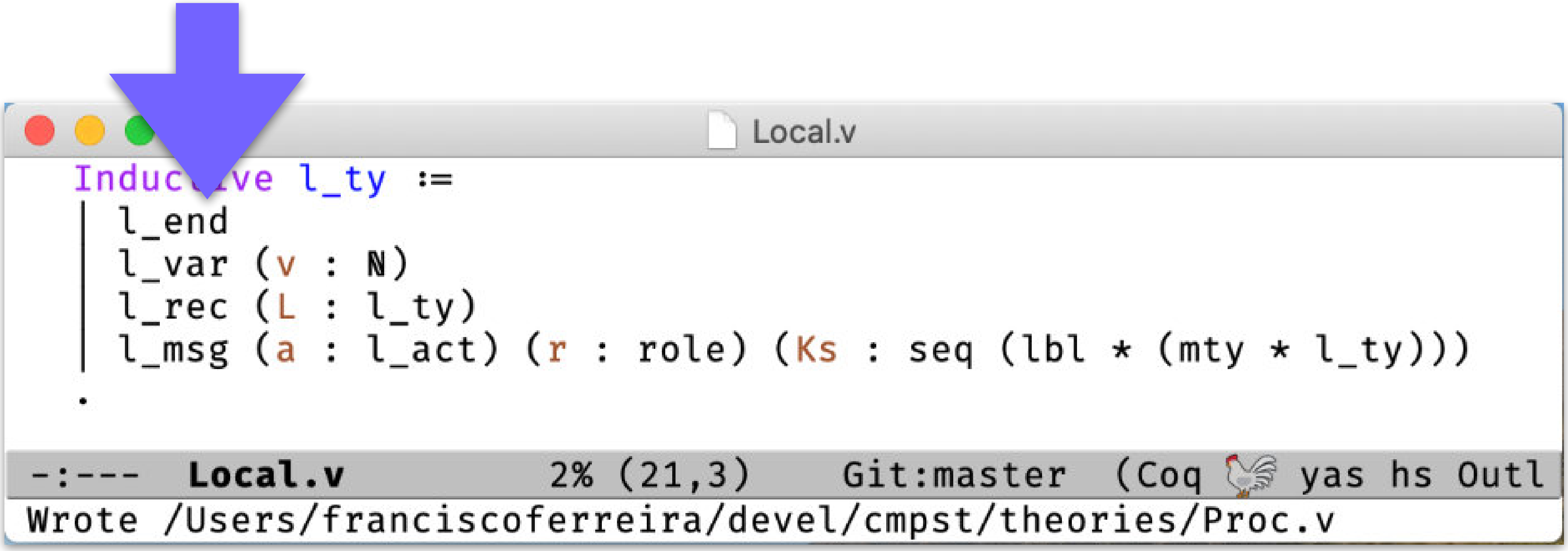
Certified code extraction

Processes into OCaml

Mechanised Meta-theory

MPST Trace equivalence

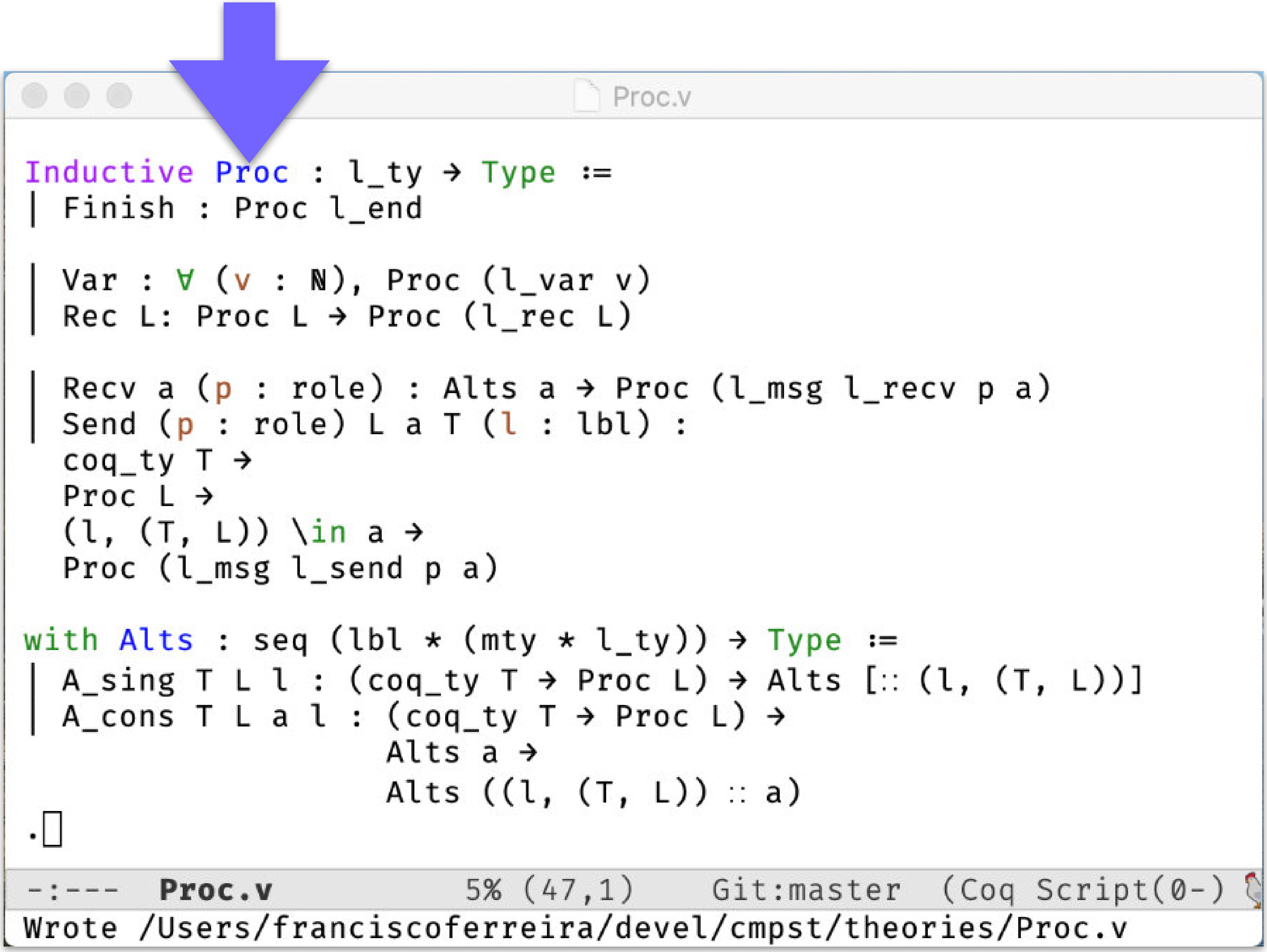
Processes : Local Types



```
  Inductive l_ty :=
  | l_end
  | l_var (v : ℕ)
  | l_rec (L : l_ty)
  | l_msg (a : l_act) (r : role) (Ks : seq (lbl * (mty * l_ty)))
  .
```

```
-:---  Local.v          2% (21,3)     Git:master   (Coq  yas hs Outl
Wrote /Users/franciscoferreira/devel/cmpst/theories/Proc.v
```

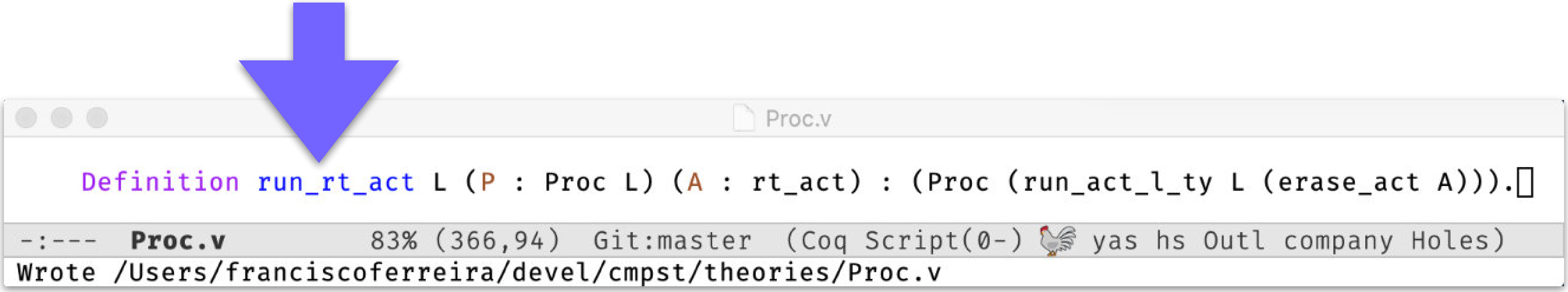
Processes



```
Inductive Proc : l_ty → Type :=
| Finish : Proc l_end

| Var : ∀ (v : ℕ), Proc (l_var v)
| Rec L: Proc L → Proc (l_rec L)

| Recv a (p : role) : Alts a → Proc (l_msg l_recv p a)
| Send (p : role) L a T (l : lbl) :
  coq_ty T →
  Proc L →
  (l, (T, L)) \in a →
  Proc (l_msg l_send p a)

with Alts : seq (lbl * (mty * l_ty)) → Type :=
| A_sing T L l : (coq_ty T → Proc L) → Alts [:: (l, (T, L))]
| A_cons T L a l : (coq_ty T → Proc L) →
                   Alts a →
                   Alts ((l, (T, L)) :: a)
.
```

```
-:---  Proc.v          5% (47,1)     Git:master  (Coq Script(0-)
Wrote /Users/franciscoferreira/devel/cmpst/theories/Proc.v
```

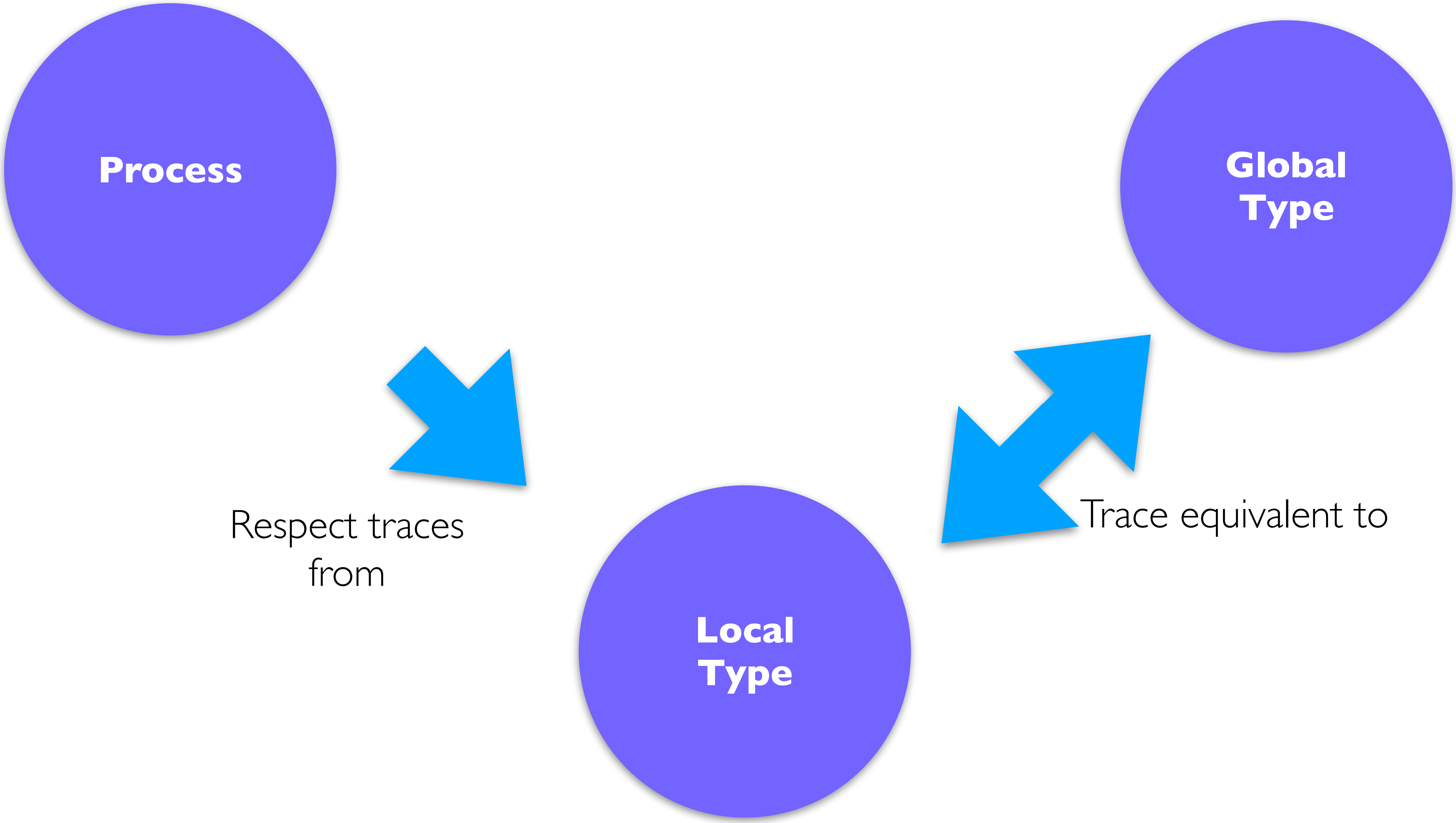
“Process Traces are Nice”



```
Definition run_rt_act L (P : Proc L) (A : rt_act) : (Proc (run_act_l_ty L (erase_act A))).
-:---  Proc.v        83% (366,94)  Git:master  (Coq Script(0-) 🐓 yas hs Outl company Holes)
Wrote /Users/franciscoferreira/devel/cmpst/theories/Proc.v
```

- Running a process preserves types by construction

From Processes to...



Process

Global Type

Respect traces from

Trace equivalent to

Local Type

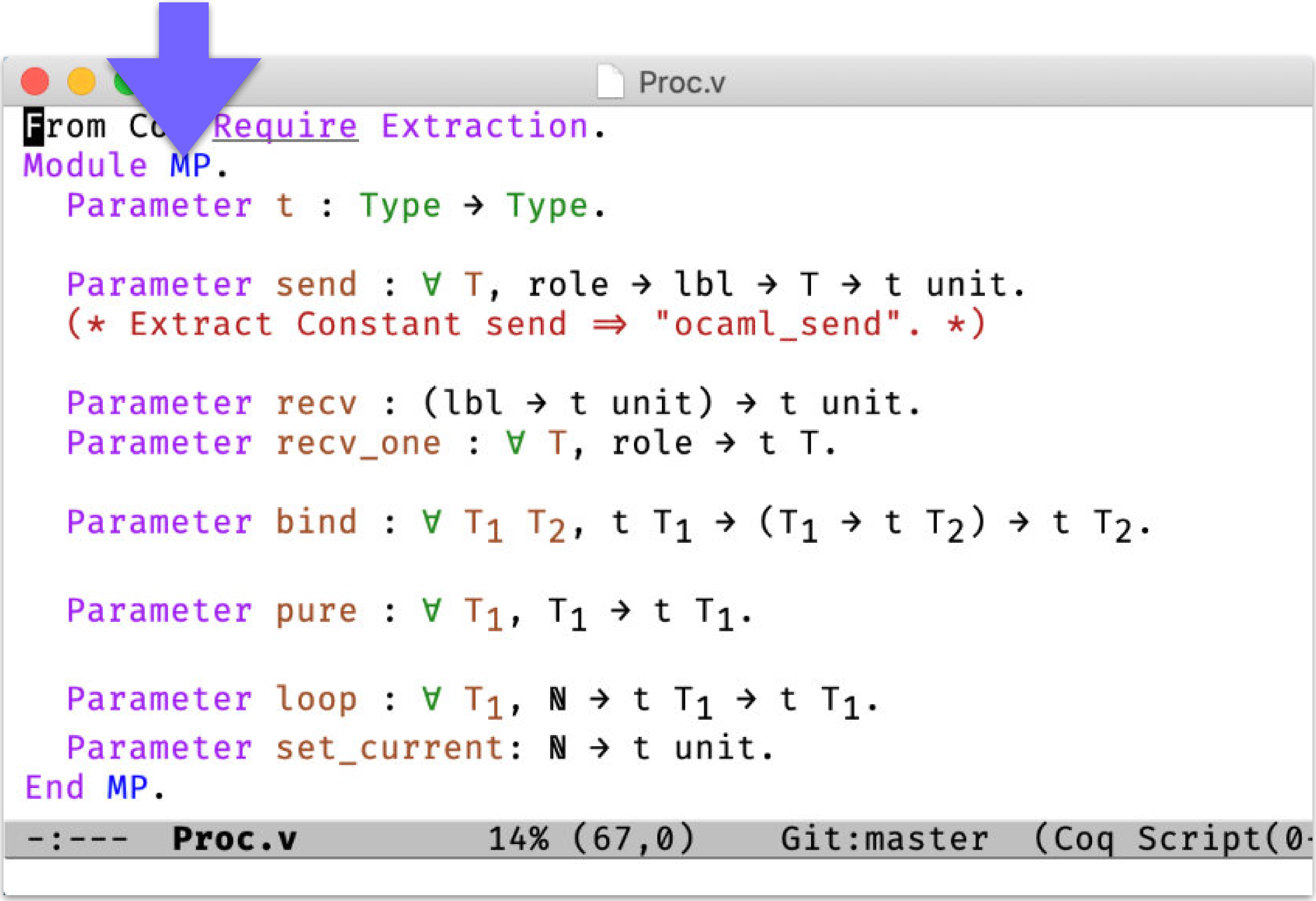
Reasoning

- A process is a term of type `Proc L.`
- The user just writes proofs on the shape of said term.
- Processes are translated into monadic computations.

Extraction of certified code

- Two aspects:
  - Generating certified OCaml code parametrised by an ambient monad.
  - Generating a certified library to handle Multiparty Session Types. Ultimately combining the *ν*Scr (a small implementation of Scribble in OCaml) to build Certified *ν*Scr.

Certified Processes



```
From Co Require Extraction.
Module MP.
  Parameter t : Type → Type.

  Parameter send : ∀ T, role → lbl → T → t unit.
  (* Extract Constant send ⇒ "ocaml_send". *)

  Parameter recv : (lbl → t unit) → t unit.
  Parameter recv_one : ∀ T, role → t T.

  Parameter bind : ∀ T₁ T₂, t T₁ → (T₁ → t T₂) → t T₂.

  Parameter pure : ∀ T₁, T₁ → t T₁.

  Parameter loop : ∀ T₁, ℕ → t T₁ → t T₁.
  Parameter set_current: ℕ → t unit.
End MP.
-:---  Proc.v         14% (67,0)     Git:master   (Coq Script(0-
```

About Proof Assistant Choice

- We chose Coq because it is powerful, well maintained, and popular in PL.

- While using it,

  😩 I wished for Isabelle’s automation and classical logic.

  😭 I cried over the loss of Agda’s dependent pattern matching and rich interaction with the system.

  🤤 As we try to get extraction to work, I envy Idris’s compiler.

If you want to know more…

- Talk to us!
- Binary Session types:
  - TACAS'20 Tool Paper: https://bit.ly/3co7KFn
  - Tech report: https://bit.ly/2ZZzAVE
  - EMTST repository: https://github.com/emtst/
- Multiparty Session Types
  - Repo: Talk to us!
  - Check *ν*Scr at: https://nuscr.github.io

Thanks for your kind attention! Questions?