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

Nobuko Yoshida¹

Imperial College London

Vasco T. Vasconcelos²

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 $\alpha$-conversion

Then we cannot distinguish: $k?(x)$ in inact and $k?(y)$ in inact
**α-conversion curse or Blessing?**

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

- The original system depends crucially on names

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

\[(\text{throw } k[k']; P_1) | (\text{catch } k(k'') \text{ in } P_2) \rightarrow P_1 \mid P_2[k'/k'']\]
**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 than 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

```plaintext
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 $A_{SC}$
- binds variable from $A_{EV}$
- binds variable from $A_{LC}$
- binds channel from $A_{CN}$
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 $\Delta$ balanced and $P \rightarrow^*$ $Q$, then $\Theta; \Gamma \vdash Q \triangleright \Delta'$ and $\Delta'$ balanced.

Is straightforward to represent:

```
Theorem SubjectReduction G P Q D:
of G P D → balanced D → P →* Q → exists D', balanced D' \∧ of G Q D'.
```
We want more from our mechanisation.
Motivating Meta-Theory

Certified tool + reasoning environment

About Processes

Reasoning

Certified code extraction

Processes into OCaml

Mechanised Meta-theory

MPST Trace equivalence
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)))
.
Processes

\begin{verbatim}
Inductive Proc : l_ty \rightarrow Type :=
| Finish : Proc l_end

| Var : \forall (v : \mathbb{N}), Proc (l_var v)
| Rec L : Proc L \rightarrow Proc (l_rec L)

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

with Alts : seq (lbl * (mty * l_ty)) \rightarrow Type :=
| A_sing T L l : (coq_ty T \rightarrow Proc L) \rightarrow Alts [:: (l, (T, L))]
| A_cons T L a l : (coq_ty T \rightarrow Proc L) \rightarrow
  Alts a \rightarrow
  Alts ((l, (T, L)) :: a)
\end{verbatim}
“Process Traces are Nice”

- Running a process preserves types by construction
From Processes to...

- Process
- Local Type
- Global Type

Respect traces from Local Type to Global Type.
Reasoning

• A process is a term of type $\text{Proc } L$.

• 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 νScr (a small implementation of Scribble in OCaml) to build Certified νScr.
Certified Processes

```
From CoC Require Extraction.
Module MP.

Parameter t : Type -> Type.

Parameter send : \forall T, role -> lbl -> T -> t unit.
   (* Extract Constant send => "ocaml_send". *)

Parameter recv : (lbl -> t unit) -> t unit.
Parameter recv_one : \forall T, role -> t T.

Parameter bind : \forall T_1 T_2, t T_1 -> (T_1 -> t T_2) -> t T_2.

Parameter pure : \forall T_1, T_1 -> t T_1.

Parameter loop : \forall T_1, \mathbb{N} -> t T_1 -> t T_1.
Parameter set_current : \mathbb{N} -> t unit.
End MP.
```
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:
- EMTST repository: https://github.com/emtst/

Multiparty Session Types
- Repo: Talk to us!
- Check νScr at: https://nuscr.github.io

Thanks for your kind attention! Questions?