Type-checking session-typed π-calculus with Coq

Uma Zalakain

University of Glasgow

Problem

Formalising session typed π-calculus in Coq

- subset (finite, no shared channels)
- strong correctness guarantees
- interesting modelling exercise
- perfect excuse to familiarise with Coq

Goal

Correctness by construction

- coq type-checks process $\iff$ process uses STs correctly
- bonus: the session types of channels are type-inferred

Ingredients

Continuation passing

- an action `A`
  consumes a channel `:A.T`
  creates a channel `:T`

Ingredients

Abstraction

- channels and messages as arguments
- variable references lifted to Coq
- no environments (only closed processes)
- no substitution lemmas

Ingredients

Parametric channel type

- opaque unforgeable channels
- indexed by session type

Ingredients

```
| PNew
  : forall (s r : SType)
  , Duality s r
  → (Message C[s] → Message C[r] → Process)
  → Process

| PInput
  : forall {m : MType} {s : SType}
  , (Message m → Message C[s] → Process)
  → Message C[? m ; s]
  → Process
```

The catch

$$\xrightarrow{x{:}C[A.T]} A(x) \xrightarrow{y{:}C[T]} A(x) \xrightarrow{z{:}C[T]} \ldots$$

Workaround

- linearity as an inductive predicate on processes
- process traversal:
  - need to construct messages of arbitrary type
  - parametrise message types
  - project messages types to the unit type
  - cannot use constructs of the metalanguage anymore
- process is linear $\iff$ process uses STs correctly

Subject reduction

$$\forall\ P\ Q : \mathit{Process},$$
$$P \to Q,\ \mathit{lin}(P) \implies \mathit{lin}(Q)$$