Formalising Subject Reduction and Progress for Multiparty Session Processes

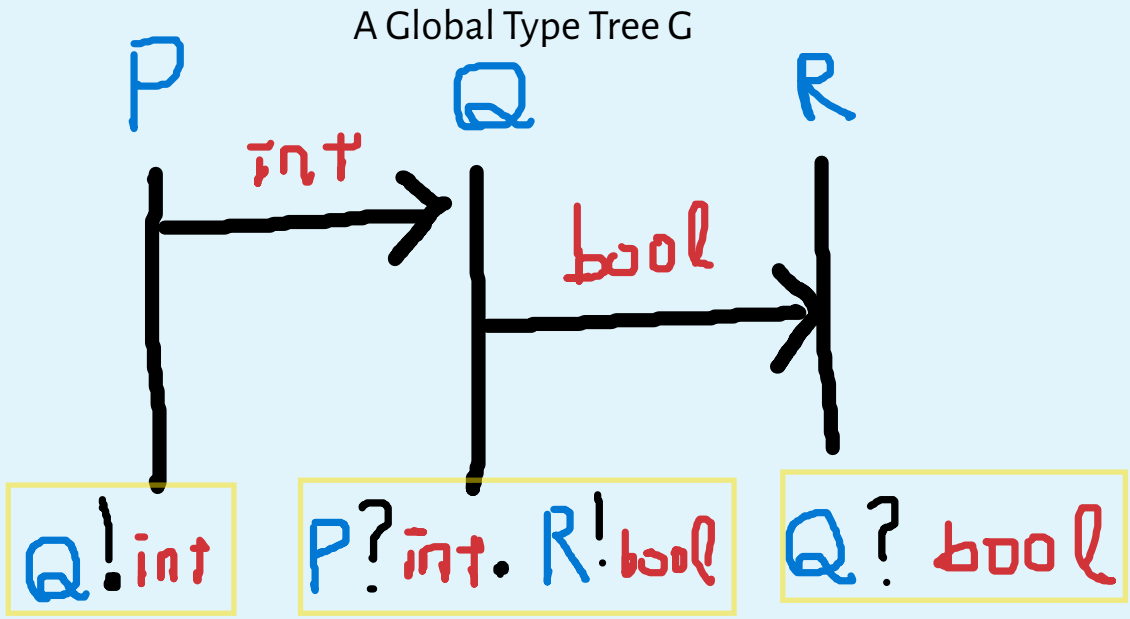
ITP 2025

Burak Ekici
Tadayoshi Kamegai
Nobuko Yoshida

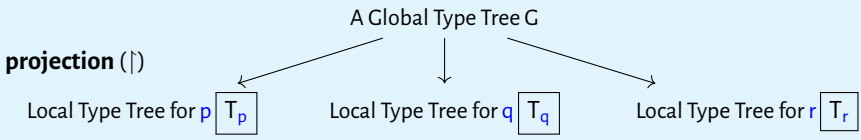
September 28, 2025

Outline

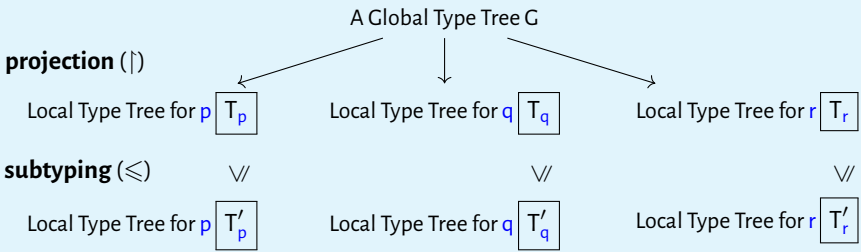
Multiparty Session Types (Top Down Approach) (Honda, NY and Carbone 2008)



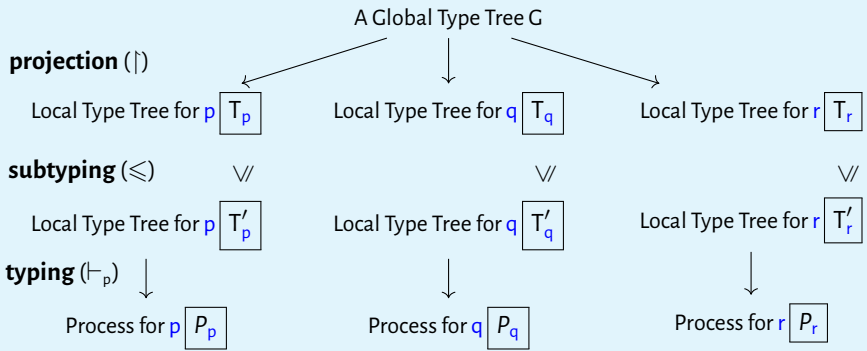
Multiparty Session Types (Top Down Approach) (Honda, NY and Carbone 2008)



Multiparty Session Types (Top Down Approach) (Honda, NY and Carbone 2008)



Multiparty Session Types (Top Down Approach) (Honda, NY and Carbone 2008)



Contributions & Challenges

- extend synchronous MPST [5] with a **mechanised proof of the non-stuck theorem**

Contributions & Challenges

- extend synchronous MPST [5] with a **mechanised proof of the non-stuck theorem**

subject reduction	well-typed sessions reduce into well-typed sessions
progress	well-typed sessions either terminate or reduce to other sessions

Contributions & Challenges

- extend synchronous MPST [5] with a **mechanised proof of the non-stuck theorem**

  subject reduction well-typed sessions reduce into well-typed sessions

  progress well-typed sessions either terminate or reduce to other sessions

  using coinductive reasoning over “type trees”

Contributions & Challenges

- extend synchronous MPST [5] with a **mechanised proof of the non-stuck theorem**

  subject reduction well-typed sessions reduce into well-typed sessions

  progress well-typed sessions either terminate or reduce to other sessions

  using coinductive reasoning over “type trees”
- integrate subtyping

Contributions & Challenges

- extend synchronous MPST [5] with a **mechanised proof of the non-stuck theorem**

  subject reduction well-typed sessions reduce into well-typed sessions

  progress well-typed sessions either terminate or reduce to other sessions

  using coinductive reasoning over "type trees"
- integrate subtyping
- decompose **balanced** type trees into finite prefixes – enabling inductive reasoning within infinite trees

Contributions & Challenges

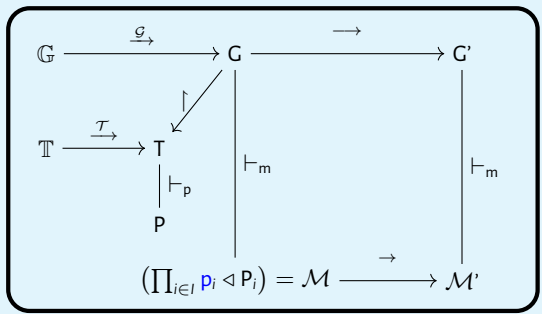
- extend synchronous MPST [5] with a **mechanised proof of the non-stuck theorem**

  subject reduction well-typed sessions reduce into well-typed sessions

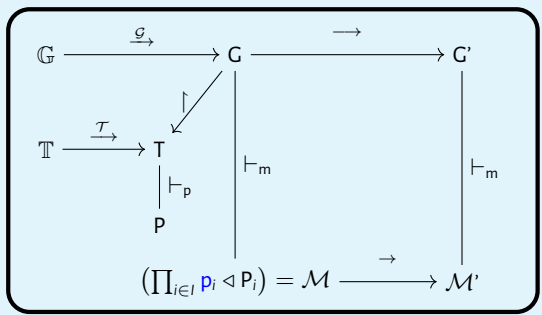
  progress well-typed sessions either terminate or reduce to other sessions

  using coinductive reasoning over "type trees"
- integrate subtyping
- decompose **balanced** type trees into finite prefixes – enabling inductive reasoning within infinite trees
- employ **finite lists** to encode, in Rcoq, continuations and branching/selections for type trees – simplifying coinductive definitions and proofs further

Design Overview



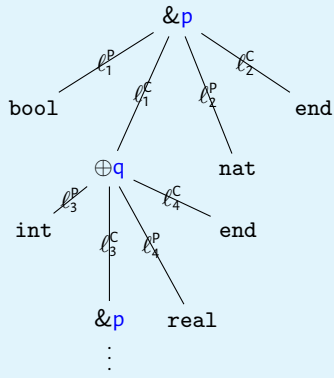
Design Overview



Outline

Session Trees (coinductive syntax)

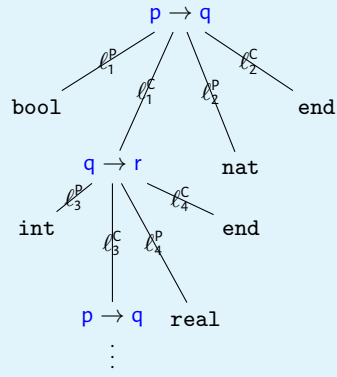
$$\mathsf{T} \quad ::= \quad \mathtt{end} \quad \mid \quad \&_{i \in I} \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \quad \mid \quad \oplus_{i \in I} \mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i$$



Session Trees (coinductive syntax)

$$\mathsf{T} \quad ::= \quad \mathtt{end} \quad \mid \quad \&_{i\in I}\, \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \quad \mid \quad \oplus_{i\in I}\, \mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i$$

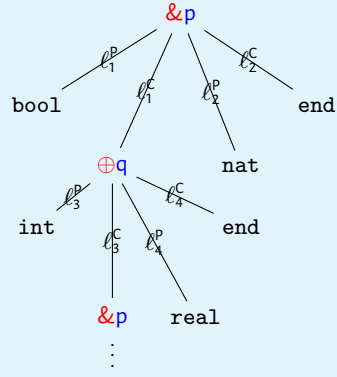
$$\mathsf{G} \quad ::= \quad \mathtt{end} \quad \mid \quad \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}$$



Session Trees (coinductive syntax)

$$\mathsf{T} \quad ::= \quad \mathtt{end} \quad \mid \quad \&_{i\in I}\, \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \quad \mid \quad \oplus_{i\in I}\, \mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i$$
$$\mathsf{G} \quad ::= \quad \mathtt{end} \quad \mid \quad \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}$$

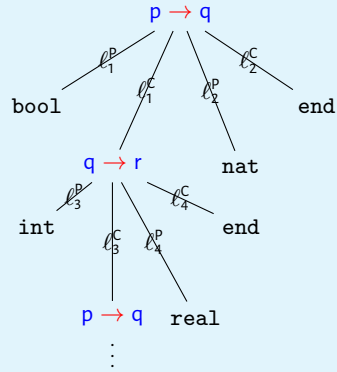
- internal nodes:
  - branching (&) or selection ($\oplus$) for local type trees



Session Trees (coinductive syntax)

$$\mathsf{T} \quad ::= \quad \mathtt{end} \quad \mid \quad \&_{i\in I}\, \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \quad \mid \quad \oplus_{i\in I}\, \mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i$$
$$\mathsf{G} \quad ::= \quad \mathtt{end} \quad \mid \quad \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}$$

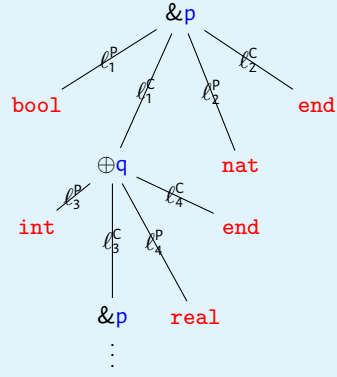
- internal nodes:
  - branching (&) or selection ($\oplus$) for local type trees
  - communication action for global type trees



Session Trees (coinductive syntax)

$$\mathsf{T} \quad ::= \quad \mathtt{end} \quad \mid \quad \&_{i \in I} \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \quad \mid \quad \oplus_{i \in I} \mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i$$
$$\mathsf{G} \quad ::= \quad \mathtt{end} \quad \mid \quad \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i \in I}$$

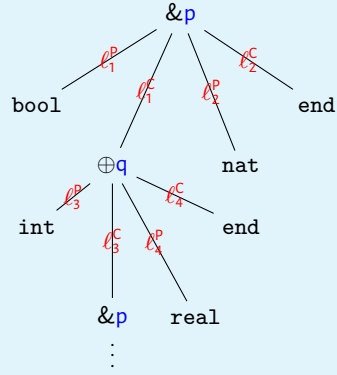
- internal nodes:
  - branching (&) or selection ($\oplus$) for local type trees
  - communication action for global type trees
- leaf nodes: payload sorts or end



Session Trees (coinductive syntax)

$$\mathsf{T} \quad ::= \quad \mathtt{end} \quad \mid \quad \&_{i\in I}\,\mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \quad \mid \quad \oplus_{i\in I}\,\mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i$$
$$\mathsf{G} \quad ::= \quad \mathtt{end} \quad \mid \quad \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}$$

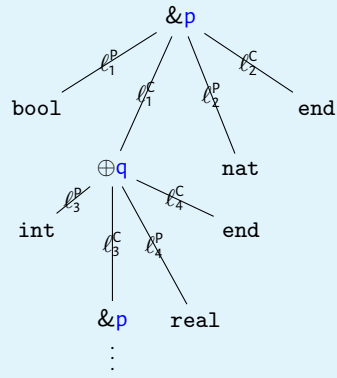
- internal nodes:
  - branching (&) or selection ($\oplus$) for local type trees
  - communication action for global type trees
- leaf nodes: payload sorts or end
- edge annotations: linking internal node to a payload (ℓ^{P}) or continuation for message (ℓ^{C})



Session Trees (coinductive syntax)

$$\mathsf{T} \quad ::= \quad \mathtt{end} \quad \mid \quad \&_{i\in I}\mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \quad \mid \quad \oplus_{i\in I}\mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i$$
$$\mathsf{G} \quad ::= \quad \mathtt{end} \quad \mid \quad \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}$$

- internal nodes:
  - branching (&) or selection ($\oplus$) for local type trees
  - communication action for global type trees
- leaf nodes: payload sorts or end
- edge annotations: linking internal node to a payload (ℓ^{P}) or continuation for message (ℓ^{C})



Global Types $\to$ Global Type Trees

represent recursive types and their unfoldings with the same tree – equi-recursive approach:

$$\frac{\mathbb{G}[\mu \mathbf{t}.\mathbb{G}/\mathbf{t}] \xrightarrow{\mathcal{G}} \mathsf{G}}{\mu \mathbf{t}.\mathbb{G} \xrightarrow{\mathcal{G}} \mathsf{G}}\ [\text{gtrans-rec}] \qquad \frac{\forall i \in I, \quad \mathbb{G}_i \xrightarrow{\mathcal{G}} \mathsf{G}_i}{\mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathbb{G}_i\}_{i\in I} \xrightarrow{\mathcal{G}} \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}}\ [\text{gtrans-send}]$$

Global Types $\to$ Global Type Trees

represent recursive types and their unfoldings with the same tree – equi-recursive approach:

$$\frac{\mathbb{G}[\mu \mathbf{t}.\mathbb{G}/\mathbf{t}] \xrightarrow{\mathcal{G}} \mathsf{G}}{\mu \mathbf{t}.\mathbb{G} \xrightarrow{\mathcal{G}} \mathsf{G}}\ [\text{gtrans-rec}] \qquad \frac{\forall i \in I, \quad \mathbb{G}_i \xrightarrow{\mathcal{G}} \mathsf{G}_i}{\mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathbb{G}_i\}_{i\in I} \xrightarrow{\mathcal{G}} \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}}\ [\text{gtrans-send}]$$

Example

$$\mathbb{G} = \mu \mathbf{t}.\mathsf{p} \to \mathsf{q} \begin{cases} \ell_1(\mathtt{bool}).\mathbf{t} \\ \ell_2(\mathtt{nat}).\mathtt{end} \end{cases}$$

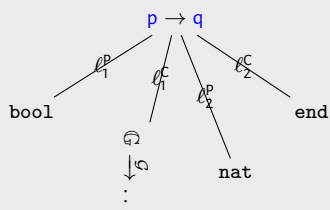
Global Types $\to$ Global Type Trees

represent recursive types and their unfoldings with the same tree – equi-recursive approach:

$$\frac{\mathbb{G}[\mu \mathbf{t}.\mathbb{G}/\mathbf{t}] \xrightarrow{\mathcal{G}} \mathsf{G}}{\mu \mathbf{t}.\mathbb{G} \xrightarrow{\mathcal{G}} \mathsf{G}}\ [\text{gtrans-rec}] \qquad \frac{\forall i \in I, \quad \mathbb{G}_i \xrightarrow{\mathcal{G}} \mathsf{G}_i}{\mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathbb{G}_i\}_{i\in I} \xrightarrow{\mathcal{G}} \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}}\ [\text{gtrans-send}]$$

Example

$$\mathbb{G} = \mu \mathbf{t}.\mathsf{p} \to \mathsf{q} \begin{cases} \ell_1(\mathtt{bool}).\mathbf{t} \\ \ell_2(\mathtt{nat}).\mathtt{end} \end{cases} \xrightarrow{\mathcal{G}}$$



Outline

Projection

extracting local type trees from global type trees:

$$\frac{\forall i \in I, G_i \upharpoonright_r T_i}{r \to q : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r \bigoplus_{i\in I} q!\ell_i(S_i).T_i} \text{ [proj-send]} \qquad \frac{\forall i \in I, G_i \upharpoonright_r T_i}{p \to r : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r \&_{i\in I}\, p?\ell_i(S_i).T_i} \text{ [proj-recv]}$$

$$\frac{r \notin \{p, q\} \quad \forall i \in I, r \in \mathtt{pt}(G_i) \quad G_i \upharpoonright_r T}{p \to q : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r T} \text{ [proj-cont]} \qquad \frac{r \notin \mathtt{pt}(G)}{G \upharpoonright_r \mathtt{end}} \text{ [proj-end]}$$

Projection

extracting local type trees from global type trees:

$$\frac{\forall i \in I, G_i \upharpoonright_r T_i}{r \to q : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r \bigoplus_{i\in I} q!\ell_i(S_i).T_i} \text{ [proj-send]} \qquad \frac{\forall i \in I, G_i \upharpoonright_r T_i}{p \to r : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r \&_{i\in I}\, p?\ell_i(S_i).T_i} \text{ [proj-recv]}$$

$$\frac{r \notin \{p, q\} \quad \forall i \in I, r \in \mathtt{pt}(G_i) \quad G_i \upharpoonright_r T}{p \to q : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r T} \text{ [proj-cont]} \qquad \frac{r \notin \mathtt{pt}(G)}{G \upharpoonright_r \mathtt{end}} \text{ [proj-end]}$$

Provable


p → q
ℓ_1^c
p → q
ℓ_1^c
p → q
ℓ_1^c
p → q
ℓ_1^c
⋮
↾r
& p
ℓ_1^c
& p
ℓ_1^c
& p
ℓ_1^c
& p
ℓ_1^c
⋮

Projection

extracting local type trees from global type trees:

$$\frac{\forall i \in I, G_i \upharpoonright_r T_i}{r \to q : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r \bigoplus_{i\in I} q!\ell_i(S_i).T_i} \text{ [proj-send]} \qquad \frac{\forall i \in I, G_i \upharpoonright_r T_i}{p \to r : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r \&_{i\in I}\, p?\ell_i(S_i).T_i} \text{ [proj-recv]}$$

$$\frac{r \notin \{p, q\} \quad \forall i \in I, r \in \mathtt{pt}(G_i) \quad G_i \upharpoonright_r T}{p \to q : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright_r T} \text{ [proj-cont]} \qquad \frac{r \notin \mathtt{pt}(G)}{G \upharpoonright_r \mathtt{end}} \text{ [proj-end]}$$

Step Relation

global type trees evolve by consuming communications:

$$\frac{\forall i \in I \qquad \exists k \in I, \ell = \ell_k}{\left(\mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}\right) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\ \mathsf{G}_k} \text{[st-eq]}$$

$$\frac{\{\mathsf{r},\mathsf{s}\} \cap \{\mathsf{p},\mathsf{q}\} = \varnothing \qquad \forall i \in I, \{\mathsf{p},\mathsf{q}\} \subseteq \mathtt{pt}(\mathsf{G}_i)}{\left(\mathsf{r} \to \mathsf{s} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}\right) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\ \left(\mathsf{r} \to \mathsf{s} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\}_{i\in I}\right)} \text{[st-neq]}$$

Step Relation

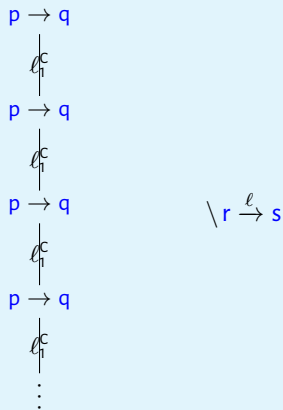
global type trees evolve by consuming communications:

$$\frac{\forall i \in I \qquad \exists k \in I, \ell = \ell_k}{\left(\mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}\right) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\ \mathsf{G}_k} \text{[st-eq]}$$

$$\frac{\{\mathsf{r},\mathsf{s}\} \cap \{\mathsf{p},\mathsf{q}\} = \varnothing \qquad \forall i \in I, \{\mathsf{p},\mathsf{q}\} \subseteq \mathtt{pt}(\mathsf{G}_i)}{\left(\mathsf{r} \to \mathsf{s} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I}\right) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\ \left(\mathsf{r} \to \mathsf{s} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\}_{i\in I}\right)} \text{[st-neq]}$$

$$\left(\mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{p} \to \mathsf{q}.\mathtt{end}\}_{i\in I}\right) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\ \left(\mathsf{r} \to \mathsf{s} : \{\ell_i.\mathtt{end}\}_{i\in I}\right)$$

Provable


p → q
ℓ_1^C
p → q
ℓ_1^C
p → q
ℓ_1^C
p → q
ℓ_1^C
⋮
\ r $\xrightarrow{\ell}$ s
p → q
ℓ_1^C
p → q
ℓ_1^C
p → q
ℓ_1^C
p → q
ℓ_1^C
⋮

Outline

Subtyping

provides flexibility to type system – use a process of type T whenever a process of type T′ is needed – $\mathsf{T} \leqslant \mathsf{T}'$

$$\frac{\forall i \in I, \quad \mathsf{S}_i \leq: \mathsf{S}'_i \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i \leqslant \bigoplus_{i\in I\cup J} \mathsf{p}!\ell_i(\mathsf{S}'_i).\mathsf{T}'_i}\ [\text{sub-out}] \qquad \frac{\forall i \in I, \quad \mathsf{S}'_i \leq: \mathsf{S}_i \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\&_{i\in I\cup J} \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \leqslant \&_{i\in I} \mathsf{p}?\ell_i(\mathsf{S}'_i).\mathsf{T}'_i}\ [\text{sub-in}]$$

Subtyping

provides flexibility to type system – use a process of type T whenever a process of type T′ is needed – $\mathsf{T} \leqslant \mathsf{T}'$

$$\frac{\forall i \in I, \quad \mathsf{S}_i \leq: \mathsf{S}'_i \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i \leqslant \bigoplus_{i\in I\cup J} \mathsf{p}!\ell_i(\mathsf{S}'_i).\mathsf{T}'_i}\ [\text{sub-out}] \qquad \frac{\forall i \in I, \quad \mathsf{S}'_i \leq: \mathsf{S}_i \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\&_{i\in I\cup J} \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \leqslant \&_{i\in I} \mathsf{p}?\ell_i(\mathsf{S}'_i).\mathsf{T}'_i}\ [\text{sub-in}]$$

Type Checking

$$\frac{\forall i \in I, \quad \Gamma, x_i : \mathsf{S}_i \vdash_\mathsf{p} \mathsf{P}_i : \mathsf{T}_i}{\Gamma \vdash_\mathsf{p} \sum_{i\in I} \mathsf{p}?\ell_i(x_i).\mathsf{P}_i : \&_{i\in I} \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i}\ [\text{tin}] \qquad \frac{\Gamma \vdash_\mathsf{s} e : \mathsf{S} \quad \Gamma \vdash_\mathsf{p} \mathsf{P} : \mathsf{T}}{\Gamma \vdash_\mathsf{p} \mathsf{p}!\ell(e).\mathsf{P} : \bigoplus \mathsf{p}!\ell(\mathsf{S}).\mathsf{T}}\ [\text{tout}]$$

$$\frac{\Gamma, \mathbf{X} : \mathsf{T} \vdash_\mathsf{p} \mathsf{P} : \mathsf{T}}{\Gamma \vdash_\mathsf{p} \mu\mathbf{X}.\mathsf{P} : \mathsf{T}}\ [\text{trec}] \qquad \frac{\Gamma \vdash_\mathsf{p} \mathsf{P} : \mathsf{T} \quad \mathsf{T} \leqslant \mathsf{T}'}{\Gamma \vdash_\mathsf{p} \mathsf{P} : \mathsf{T}'}\ [\text{tsub}] \qquad \frac{\Gamma \vdash_\mathsf{s} e : \texttt{bool} \quad \Gamma \vdash_\mathsf{p} \mathsf{P}_1 : \mathsf{T} \quad \Gamma \vdash_\mathsf{p} \mathsf{P}_2 : \mathsf{T}}{\Gamma \vdash_\mathsf{p} \text{if } e \text{ then } \mathsf{P}_1 \text{ else } \mathsf{P}_2 : \mathsf{T}}\ [\text{tite}]$$

$$\frac{\forall i \in I, \quad \mathsf{G}\upharpoonright_{\mathsf{p}_i} \mathsf{T}_i \quad \vdash_\mathsf{p} \mathsf{P}_i : \mathsf{T}_i \quad \mathtt{pt}(\mathsf{G}) \subseteq \{\mathsf{p}_i \mid i \in I\}}{\vdash_\mathsf{m} \prod_{i\in I} \mathsf{p}_i \lhd \mathsf{P}_i : \mathsf{G}}\ [\text{tsess}]$$

Processes and Multiparty Sessions (syntax)

$$\begin{array}{lcl} \mathtt{P} & ::= & \mathtt{p}!\ell(e).\mathtt{P} \mid \sum_{i\in I} \mathtt{p}?\ell_i(x_i).\mathtt{P}_i \mid \mathtt{if}\ \mathtt{e}\ \mathtt{then}\ \mathtt{P}\ \mathtt{else}\ \mathtt{P} \mid \mu\mathbf{X}.\mathtt{P} \mid \mathbf{X} \mid \mathbf{0} \\ \mathcal{M} & ::= & \mathtt{p} \triangleleft \mathtt{P} \mid \mathcal{M} \mid \mathcal{M} \end{array}$$

Processes and Multiparty Sessions (pre-congruence and reduction)

$$\frac{}{\mathtt{p} \triangleleft \mu\mathbf{X}.\mathtt{P} \mid \mathcal{M} \Rrightarrow \mathtt{p} \triangleleft \mathtt{P}[\mu\mathbf{X}.\mathtt{P}/\mathbf{X}] \mid \mathcal{M}}\ [\text{po-unf}]$$

$$\frac{\forall i \in I \quad j \in I \quad e \downarrow v}{\mathtt{p} \triangleleft \sum_{i\in I} \mathtt{q}?\ell_i(x_i).\mathtt{P}_i \mid \mathtt{q} \triangleleft \mathtt{p}!\ell_j(e).\mathtt{Q} \mid \mathcal{M} \longrightarrow \mathtt{p} \triangleleft \mathtt{P}_j[v/x_j] \mid \mathtt{q} \triangleleft \mathtt{Q} \mid \mathcal{M}}\ [\text{r-comm}]$$

$$\frac{\mathcal{M}_1' \Rrightarrow \mathcal{M}_1 \quad \mathcal{M}_1 \longrightarrow \mathcal{M}_2 \quad \mathcal{M}_2 \Rrightarrow \mathcal{M}_2'}{\mathcal{M}_1' \longrightarrow \mathcal{M}_2'}\ [\text{r-struct}]$$

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$ does not hold!

some prior work invalidating subject reduction [1, 5, 11, 6, 7]

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$ does not hold!

some prior work invalidating subject reduction [1, 5, 11, 6, 7]

Counter Example

let a process $P = p?\ell(x).p!\ell'(x).\mathbf{X}$

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$ does not hold!

some prior work invalidating subject reduction [1, 5, 11, 6, 7]

Counter Example

let a process $P = p?\ell(x).p!\ell'(x).\mathbf{X}$

have $\vdash_p P[\mu\mathbf{X}.P/\mathbf{X}] : T$

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$ does not hold!

some prior work invalidating subject reduction [1, 5, 11, 6, 7]

Counter Example

let a process $P = p?\ell(x).p!\ell'(x).\mathbf{X}$

have $\vdash_p P[\mu\mathbf{X}.P/\mathbf{X}] : T$ where

$T = p?\ell(\texttt{bool}).p!\ell'(\texttt{bool}).p?\ell(\texttt{nat}).p!\ell'(\texttt{nat}).T$

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$ does not hold!

some prior work invalidating subject reduction [1, 5, 11, 6, 7]

Counter Example

let a process $P = p?\ell(x).p!\ell'(x).\mathbf{X}$

have $\vdash_p P[\mu\mathbf{X}.P/\mathbf{X}] : T$ where

$T = p?\ell(\texttt{bool}).p!\ell'(\texttt{bool}).p?\ell(\texttt{nat}).p!\ell'(\texttt{nat}).T$

however $\not\vdash_p \mu\mathbf{X}.P : T$

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$ does not hold!

some prior work invalidating subject reduction [1, 5, 11, 6, 7]

Counter Example

let a process $P = p?\ell(x).p!\ell'(x).\mathbf{X}$

have $\vdash_p P[\mu\mathbf{X}.P/\mathbf{X}] : T$ where

$T = p?\ell(\texttt{bool}).p!\ell'(\texttt{bool}).p?\ell(\texttt{nat}).p!\ell'(\texttt{nat}).T$

however $\not\vdash_p \mu\mathbf{X}.P : T$

a solution is to disable fold-back identities

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$ does not hold!

some prior work invalidating subject reduction [1, 5, 11, 6, 7]

Counter Example

let a process $P = p?\ell(x).p!\ell'(x).\mathbf{X}$

have $\vdash_p P[\mu\mathbf{X}.P/\mathbf{X}] : T$ where

$T = p?\ell(\texttt{bool}).p!\ell'(\texttt{bool}).p?\ell(\texttt{nat}).p!\ell'(\texttt{nat}).T$

however $\not\vdash_p \mu\mathbf{X}.P : T$

a solution is to disable fold-back identities ($\Rrightarrow$ handles that)

Congruence (violates subject reduction)

Assume $\Gamma \vdash_p P : T$ and $P \equiv Q$. Then we have $\Gamma \vdash_p Q : T$ does not hold!

some prior work invalidating subject reduction [1, 5, 11, 6, 7]

Counter Example

let a process $P = p?\ell(x).p!\ell'(x).\mathbf{X}$

have $\vdash_p P[\mu\mathbf{X}.P/\mathbf{X}] : T$ where

$T = p?\ell(\texttt{bool}).p!\ell'(\texttt{bool}).p?\ell(\texttt{nat}).p!\ell'(\texttt{nat}).T$

however $\not\vdash_p \mu\mathbf{X}.P : T$

a solution is to disable fold-back identities ($\Rrightarrow$ handles that)

imported by some recently published work [13, 2]

Outline

Balancedness

G is balanced if, for every subtree G' of G, whenever $p \in \mathtt{pt}(G')$, $\exists\, k \in \mathbb{N}$ such that

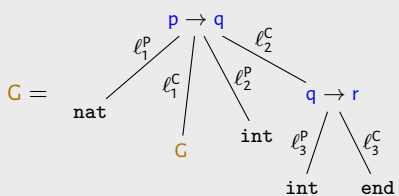
1. For all paths γ of length k from the root of G', $p \in \gamma$
2. For all paths γ from the root of G' that end at a leaf ($\mathtt{end}$), $p \in \gamma$

Balancedness

G is balanced if, for every subtree G' of G, whenever $p \in \mathtt{pt}(G')$, $\exists\, k \in \mathbb{N}$ such that

1. For all paths γ of length k from the root of G', $p \in \gamma$
2. For all paths γ from the root of G' that end at a leaf (`end`), $p \in \gamma$

Example (an unbalanced tree)

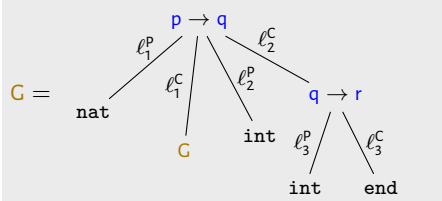


Balancedness

G is balanced if, for every subtree G' of G, whenever $p \in \mathtt{pt}(G')$, $\exists\, k \in \mathbb{N}$ such that

1. For all paths γ of length k from the root of G', $p \in \gamma$
2. For all paths γ from the root of G' that end at a leaf (`end`), $p \in \gamma$

Example (an unbalanced tree)



Well-formedness

Global type tree G is *well-formed* if $\exists$ global type $\mathbb{G}$,

1. recursion is guarded
2. continuations non-empty and non-$\bot$
3. $\mathbb{G} \xrightarrow{\mathcal{G}} G$
4. G is balanced

Definition (Grafting)

$$\Gamma_G \quad ::= \quad p \to q : \{\ell_i(S_i).\Gamma_{G_i}\}_{i \in I} \quad | \quad [\,]_i$$

Definition (Grafting)

$$\Gamma_G \quad ::= \quad p \to q : \{\ell_i(S_i).\Gamma_{G_i}\}_{i \in I} \quad | \quad [\,]_i$$

- construct a global tree G by filling *all holes* in an input context Γ_G with elements of a list of global type trees L
- denoted $G = \Gamma_G[L]$

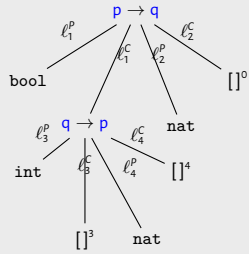
Definition (Grafting)

$$\Gamma_G \quad ::= \quad p \to q : \{\ell_i(S_i).\Gamma_{G_i}\}_{i \in I} \quad | \quad []_i$$

- construct a global tree G by filling *all holes* in an input context Γ_G with elements of a list of global type trees L
- denoted $G = \Gamma_G[L]$

Example (Grafting)

$\Gamma_G =$



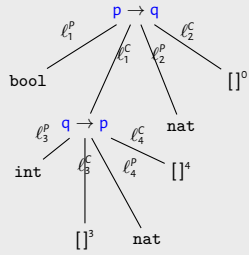
Definition (Grafting)

$$\Gamma_G \quad ::= \quad p \to q : \{\ell_i(S_i).\Gamma_{G_i}\}_{i \in I} \quad | \quad []_i$$

- construct a global tree G by filling *all holes* in an input context Γ_G with elements of a list of global type trees L
- denoted $G = \Gamma_G[L]$

Example (Grafting)

$\Gamma_G =$



$L = [G_0, G_1, G_2, G_3, G_4, G_5]$

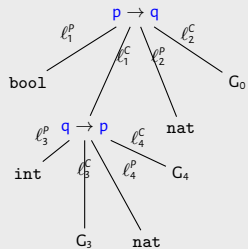
Definition (Grafting)

$$\Gamma_G \quad ::= \quad p \to q : \{\ell_i(S_i).\Gamma_{G_i}\}_{i \in I} \quad | \quad [\,]_i$$

- construct a global tree G by filling *all holes* in an input context Γ_G with elements of a list of global type trees L
- denoted $G = \Gamma_G[L]$

Example (Grafting)

$\Gamma_G[L] =$



$L = [G_0, G_1, G_2, G_3, G_4, G_5]$

Lemma (grafting)

If $\texttt{p} \in_g \texttt{pt(G)}$. Then,

- $\exists$ `L, Gl` such that `G = Gl[L]` with $\texttt{p} \notin_h \texttt{pt(Gl)}$,
- elements filling a hole in `Gl` from `L` is of $\texttt{p} \to \texttt{q lsg}$, $\texttt{q} \to \texttt{p lsg}$ or `end`

Lemma (grafting)

If `p` $\in_g$ `pt(G)` . Then,

- $\exists$ `L, Gl` such that `G = Gl[L]` with `p` $\notin_h$ `pt(Gl)` ,
- elements filling a hole in `Gl` from `L` is of `p` $\to$ `q lsg` , `q` $\to$ `p lsg` or `end`

Notation

Represent continuations of global and local type trees as *option lists* of *sort-tree* pairs: `p` $\to$ `q L` and $\oplus$`p! L`

Lemma (preservation of projection under reduction)

If we have

- $\texttt{G} \upharpoonright_{\texttt{p}} (\oplus \texttt{q! l}_1)$, $\texttt{G} \upharpoonright_{\texttt{q}} (\&\texttt{p? l}_2)$, $(\pi_{\texttt{T}}\ \texttt{l}_1)_{\texttt{n}} = \texttt{T}$, $(\pi_{\texttt{T}}\ \texttt{l}_2)_{\texttt{n}} = \texttt{T'}$, and $\texttt{G} \setminus \texttt{p} \xrightarrow{\texttt{n}} \texttt{q G'}$.

Then,

- $\texttt{G'} \upharpoonright_{\texttt{p}} \texttt{T}$ and $\texttt{G'} \upharpoonright_{\texttt{q}} \texttt{T'}$

Lemma (preservation of projection under reduction)

If we have

- $\texttt{G} \upharpoonright_{\texttt{p}} (\oplus\texttt{q!}\ \texttt{l}_1)$, $\texttt{G} \upharpoonright_{\texttt{q}} (\&\texttt{p?}\ \texttt{l}_2)$, $(\pi_{\texttt{T}}\ \texttt{l}_1)_{\texttt{n}} = \texttt{T}$, $(\pi_{\texttt{T}}\ \texttt{l}_2)_{\texttt{n}} = \texttt{T'}$, and $\texttt{G} \setminus \texttt{p} \xrightarrow{\texttt{n}} \texttt{q}\ \texttt{G'}$.

Then,

- $\texttt{G'} \upharpoonright_{\texttt{p}} \texttt{T}$ and $\texttt{G'} \upharpoonright_{\texttt{q}} \texttt{T'}$

Lemma (consumption from projection and subtyping)

If

- $\texttt{G} \upharpoonright_{\texttt{p}} (\oplus\texttt{q!}\ \texttt{l}_1)$, $\texttt{G} \upharpoonright_{\texttt{q}} (\&\texttt{p?}\ \texttt{l}_2)$,
- $\&\texttt{p?}\ \texttt{xs} \leqslant \&\texttt{p?}\ \texttt{l}_2$ with $(\texttt{xs})_{\texttt{n}} = (\texttt{s'},\texttt{T'})$, and
- $\oplus\texttt{q!}\ (+[\texttt{n}]\ (\texttt{s}, \texttt{T})) \leqslant \oplus\texttt{q!}\ \texttt{l}_1$.

Then,

- $\exists\ \texttt{G'}$ such that $\texttt{G} \setminus \texttt{p} \xrightarrow{\texttt{n}} \texttt{q}\ \texttt{G'}$.

Theorem (subject reduction)

Have

- $\vdash M : G$ and $M \longrightarrow M'$

Then

- $\exists\ G'$ such that $\vdash M' : G'$ and $G \longrightarrow G'$

Theorem (subject reduction)

Have

- $\vdash M : G$ and $M \longrightarrow M'$

Then

- $\exists$ `G'` such that $\vdash M' : G'$ and $G \longrightarrow G'$

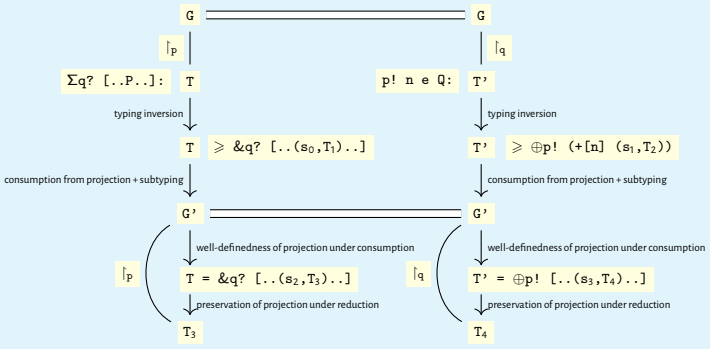
Proof sketch ([r-comm])

$$\begin{cases} (H) & \vdash \texttt{(p} \triangleleft \texttt{\&q? L | q} \triangleleft \texttt{} \oplus \texttt{p! n e Q | M): G}\,, \\ (Hn) & \texttt{L}_\texttt{n} \texttt{ = Some P}\,. \end{cases}$$

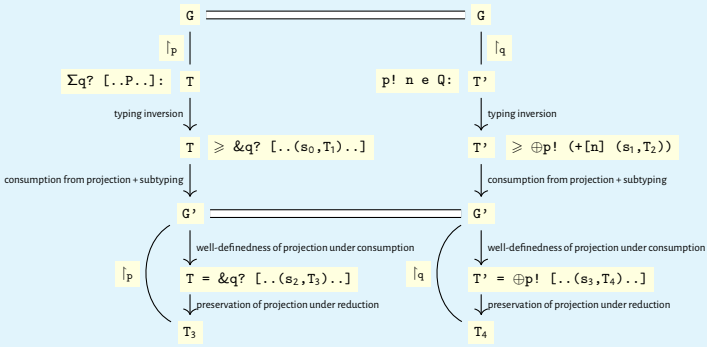
with `e` reduces into the value `v`, and the goal looks like

$$\begin{cases} (G_1) & \exists \texttt{ G'} \text{ such that } \vdash \texttt{(p} \triangleleft \texttt{ P[v/0] | q} \triangleleft \texttt{ Q | M): G'} \\ (G_2) & \texttt{G} \longrightarrow \texttt{G'}\,. \end{cases}$$

Proof sketch ([r-comm] rule)

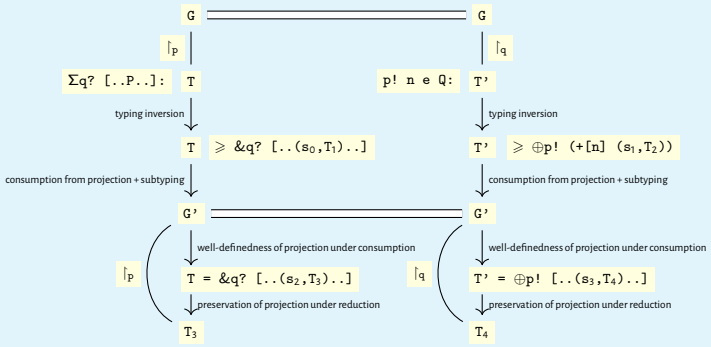


Proof sketch ([r-comm] rule)



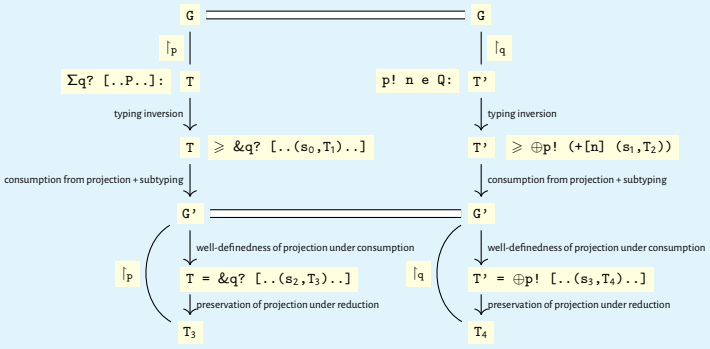
- typing inversion additionally gives $\vdash$ `Q: T2` , $\vdash$ `v: s` with $s \preceq s_2$ and $0 : s_0 \vdash$ `P: T1`

Proof sketch ([r-comm] rule)



- typing inversion additionally gives $\vdash$ `Q: T2` , $\vdash$ `v: s` with $s \preceq s_2$ and $0 : s_0 \vdash$ `P: T1`
- plug `G'` as the existential global tree

Proof sketch ([r-comm] rule)



- typing inversion additionally gives $\vdash$ `Q: T2` , $\vdash$ `v: s` with $s \preceq s_2$ and `0` : $s_0 \vdash$ `P: T1`
- plug `G'` as the existential global tree
- supposed to show $\vdash$ `P[v/0]: T1` , $\vdash$ `Q: T4` and $\vdash$ `M: G'`

Lemma (canonical forms for processes and sessions)

- Given $\vdash$ `M: (p → q xs)` . Then,
  - $\exists$ `M'` such that `M` $\Rightarrow$ `M'` , and `M'` is of `p ◁ P | q ◁ Q | M''` or `p ◁ P | q ◁ Q` form
- Given $\vdash$ `M: end` . Then,
  - $\exists$ `M'` such that `M` $\Rightarrow$ `M'` , and every process in `M'` is **0**

Lemma (canonical forms for processes and sessions)

- Given $\vdash$ `M: (p → q xs)` . Then,
  - $\exists$ `M'` such that `M` $\Rrightarrow$ `M'` , and `M'` is of `p ◁ P | q ◁ Q | M''` or `p ◁ P | q ◁ Q` form
- Given $\vdash$ `M: end` . Then,
  - $\exists$ `M'` such that `M` $\Rrightarrow$ `M'` , and every process in `M'` is **0**

Theorem (progress)

If $\vdash$ `M: G` , then $\exists$ `M'` such that `M` $\longrightarrow$ `M'` , or both `M` $\Rrightarrow$ `M'` and every process in `M'` is **0**

Lemma (canonical forms for processes and sessions)

- Given $\vdash$ `M: (p → q xs)` . Then,
  - $\exists$ `M'` such that `M` $\Rrightarrow$ `M'` , and `M'` is of `p ◁ P | q ◁ Q | M''` or `p ◁ P | q ◁ Q` form
- Given $\vdash$ `M: end` . Then,
  - $\exists$ `M'` such that `M` $\Rrightarrow$ `M'` , and every process in `M'` is **0**

Theorem (progress)

If $\vdash$ `M: G` , then $\exists$ `M'` such that `M` $\longrightarrow$ `M'` , or both `M` $\Rrightarrow$ `M'` and every process in `M'` is **0**

Proof.

by case split on `G` – matching it with canonical forms of `M`

Lemma (canonical forms for processes and sessions)

- Given $\vdash$ `M: (p → q xs)` . Then,
  - $\exists$ `M'` such that `M` $\Rrightarrow$ `M'` , and `M'` is of `p ◁ P | q ◁ Q | M''` or `p ◁ P | q ◁ Q` form
- Given $\vdash$ `M: end` . Then,
  - $\exists$ `M'` such that `M` $\Rrightarrow$ `M'` , and every process in `M'` is **0**

Theorem (progress)

If $\vdash$ `M: G` , then $\exists$ `M'` such that `M` $\longrightarrow$ `M'` , or both `M` $\Rrightarrow$ `M'` and every process in `M'` is **0**

Proof.

by case split on `G` – matching it with canonical forms of `M`

Theorem (non-stuck)

If $\vdash$ `M: G` , then `M` does not get stuck.

Outline

Approach

- usual inductive declarations for session types

```
Inductive global  : Type ≜
  | g_end : global
  | g_var : nat → global
  | g_send: part → part → list(option(sort*global)) → global
  | g_rec : global → global.
```

Approach

- usual inductive declarations for session types

```
Inductive global  : Type ≜
  | g_end : global
  | g_var : nat → global
  | g_send: part → part → list(option(sort*global)) → global
  | g_rec : global → global.
```

- positive coinductive type declarations

```
CoInductive gtt: Type ≜
  | gtt_end : gtt
  | gtt_send: part → part → list(option(sort*gtt)) → gtt.
```

Approach

- usual inductive declarations for session types

```
Inductive global  : Type ≜
  | g_end : global
  | g_var : nat → global
  | g_send: part → part → list(option(sort*global)) → global
  | g_rec : global → global.
```

- positive coinductive type declarations

```
CoInductive gtt: Type ≜
  | gtt_end : gtt
  | gtt_send: part → part → list(option(sort*gtt)) → gtt.
```

Parametrised Coinduction

- technique to aid coinductive goals in proof assistants

Parametrised Coinduction

- technique to aid coinductive goals in proof assistants
- parametrised coinduction := ordinary coinduction parametrised by a relation `R` — *knowledge accumulator* — to keep track on visited proof states in a coinductive proof

Parametrised Coinduction

- technique to aid coinductive goals in proof assistants
- parametrised coinduction := ordinary coinduction parametrised by a relation `R` — *knowledge accumulator* — to keep track on visited proof states in a coinductive proof
- parametrised greatest cofixpoint := ordinary greatest cofixpoint extended by `R`

Parametrised Coinduction

- technique to aid coinductive goals in proof assistants
- parametrised coinduction := ordinary coinduction parametrised by a relation `R` — *knowledge accumulator* — to keep track on visited proof states in a coinductive proof
- parametrised greatest cofixpoint := ordinary greatest cofixpoint extended by `R`
- obviously, when `R` is empty

ordinary greatest cofixpoint = parametrised greatest cofixpoint

Paco (Hur et al. [9, 14])

- a Coq library that implements parametrised coinduction

Paco (Hur et al. [9, 14])

- a Coq library that implements parametrised coinduction
- *greatest cofixpoint of the (parametrised) least fixpoint* technique – `paco`$_n$ construct

```
Inductive gttT (R : global → gtt → Prop) : global → gtt → Prop ≜
  | ...
  | gttT_rec: ∀ G Q G', subst_global 0 0 (g_rec G) G Q → R Q G' → gttT R (g_rec G) G'.

Definition gttTC G G' ≜ paco2 gttT bot2 G G'.
```

Paco (Hur et al. [9, 14])

- a Coq library that implements parametrised coinduction
- *greatest cofixpoint of the (parametrised) least fixpoint* technique – `paco`$_n$ construct

```
Inductive gttT (R : global → gtt → Prop) : global → gtt → Prop ≜
  | ...
  | gttT_rec: ∀ G Q G', subst_global 0 0 (g_rec G) G Q → R Q G' → gttT R (g_rec G) G'.

Definition gttTC G G' ≜ paco2 gttT bot2 G G'.
```

- `gttTC` is the greatest cofixpoint of the the generating function/relation `gttT` alas it is *monotone* – entirely non-decreasing (Knaster-Tarski semantics)

Paco (Hur et al. [9, 14])

- a Coq library that implements parametrised coinduction
- *greatest cofixpoint of the (parametrised) least fixpoint* technique – `paco`$_n$ construct

```
Inductive gttT (R : global → gtt → Prop) : global → gtt → Prop ≜
  | ...
  | gttT_rec: ∀ G Q G', subst_global 0 0 (g_rec G) G Q → R Q G' → gttT R (g_rec G) G'.

Definition gttTC G G' ≜ paco2 gttT bot2 G G'.
```

- `gttTC` is the greatest cofixpoint of the the generating function/relation `gttT` alas it is *monotone* – entirely non-decreasing (Knaster-Tarski semantics)
- `bot2` is the empty relation of arity 2 – noting in *knowledge accumulator* to start with; ordinary cofixpoint

Paco (Hur et al. [9, 14])

- a Coq library that implements parametrised coinduction
- *greatest cofixpoint of the (parametrised) least fixpoint* technique – `paco`$_n$ construct

```
Inductive gttT (R : global → gtt → Prop) : global → gtt → Prop ≜
  | ...
  | gttT_rec: ∀ G Q G', subst_global 0 0 (g_rec G) G Q → R Q G' → gttT R (g_rec G) G'.

Definition gttTC G G' ≜ paco2 gttT bot2 G G'.
```

- `gttTC` is the greatest cofixpoint of the the generating function/relation `gttT` alas it is *monotone* – entirely non-decreasing (Knaster-Tarski semantics)
- `bot2` is the empty relation of arity 2 – noting in *knowledge accumulator* to start with; ordinary cofixpoint
- coinductive proof structure in Rocq: expand the accumulator and close the goal if it is therein

Approach (use of `list` in `gtt` (cont'd))

- use of lists $\longrightarrow$ relaxing related definitions and (universal) proofs in Rocq
  - suitable with parametrised corecursive definitions with paco (translation, projection, consumption, subtyping)
  - allows for inductive reasoning within coinductive proofs

Approach (use of `list` in `gtt` (cont'd))

- use of lists $\longrightarrow$ relaxing related definitions and (universal) proofs in Rocq
  - suitable with parametrised corecursive definitions with paco (translation, projection, consumption, subtyping)
  - allows for inductive reasoning within coinductive proofs
- existential properties are problematic

```
Lemma translate: ∀ (g: global), ∃ (G: gtt), G = gttT g.
```

is not provable

Approach (use of `list` in `gtt` (cont'd))

- use of lists → relaxing related definitions and (universal) proofs in Rocq
  - suitable with parametrised corecursive definitions with paco (translation, projection, consumption, subtyping)
  - allows for inductive reasoning within coinductive proofs
- existential properties are problematic

```
Lemma translate: ∀ (g: global), ∃ (G: gtt), G = gttT g.
```

  is not provable

- use `colists` or function types in `gtt`

```
CoInductive gtt: Type ≜
  | gtt_end : gtt
  | gtt_send: part → part → (label -> option(sort*gtt)) → gtt.
```

  allows for implementing "`translate`" as a cofixpoint

Theorem

if projecting a *well-formed* tree G onto a participant p results in trees T_1 and T_2, then $T_1 = T_2$

Theorem

if projecting a *well-formed* tree G onto a participant p results in trees T_1 and T_2, then $T_1 = T_2$

Approach (coinductive extensionality (cont'd))

define a bisimulation $\sim$ over local type trees T (structural sameness) and treat it as the Leibniz equality "="

coinductive extensionality: $T_1 \sim T_2 \implies T_1 = T_2$

Theorem

if projecting a *well-formed* tree G onto a participant p results in trees T_1 and T_2, then $T_1 = T_2$

Approach (coinductive extensionality (cont'd))

define a bisimulation $\sim$ over local type trees T (structural sameness) and treat it as the Leibniz equality "="

$$\text{coinductive extensionality:} \quad T_1 \sim T_2 \implies T_1 = T_2$$

justifying soundness:

- coinductive types can be implemented as "function types" in Rocq

Theorem

if projecting a *well-formed* tree G onto a participant p results in trees T_1 and T_2, then $T_1 = T_2$

Approach (coinductive extensionality (cont'd))

define a bisimulation $\sim$ over local type trees T (structural sameness) and treat it as the Leibniz equality "="

$$\text{coinductive extensionality:} \quad T_1 \sim T_2 \implies T_1 = T_2$$

justifying soundness:

- coinductive types can be implemented as "function types" in Rocq
- coinductive types modulo coinductive extensionality is equivalent to function types with functional extensionality

Mechanisation Effort

- $\sim$16K lines of Rocq code

Mechanisation Effort

- $\sim$16K lines of Rocq code
- 341 lemmata

Mechanisation Effort

- $\sim$16K lines of Rocq code
- 341 lemmata
- 117 definitions (9 coinductive)

Mechanisation Effort

- $\sim$16K lines of Rocq code
- 341 lemmata
- 117 definitions (9 coinductive)
- the use of classical reasoning to conduct case analysis over coinductively defined predicates

Mechanisation Effort

- $\sim$16K lines of Rocq code
- 341 lemmata
- 117 definitions (9 coinductive)
- the use of classical reasoning to conduct case analysis over coinductively defined predicates
- accessible at: `https://github.com/Apiros3/smpst-sr-smer`

Related & Future Work

Related:

1. Zooid [3]: certified multiparty communication in Rocq, ensuring deadlock-free, protocol-compliant execution
2. Multris [8]: Iris framework for local protocol consistency in multiparty concurrency, no global type guarantees
3. MPGV [10]: Linear λ-calculus + MPST; deadlock-free, with progress & preservation in separation logic of Iris
4. Ekici and Yoshida [4] formalise, in Rcoq, subtyping properties in asynchronous multiparty communication
5. Tirore [12] in his PhD thesis formalises *subject reduction* in Rocq for the multiparty session π-calculus

Related & Future Work

Related:

1. Zooid [3]: certified multiparty communication in Rocq, ensuring deadlock-free, protocol-compliant execution
2. Multris [8]: Iris framework for local protocol consistency in multiparty concurrency, no global type guarantees
3. MPGV [10]: Linear λ-calculus + MPST; deadlock-free, with progress & preservation in separation logic of Iris
4. Ekici and Yoshida [4] formalise, in Rcoq, subtyping properties in asynchronous multiparty communication
5. Tirore [12] in his PhD thesis formalises *subject reduction* in Rocq for the multiparty session π-calculus

Future:

1. incorporating **full merging** into projections
2. full formal proof of **liveness** for synchronous MPST

Thanks! & Questions?

https://github.com/Apiros3/smpst-sr-smer

References

[1] Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou.
Designing asynchronous multiparty protocols with crash-stop failures.
In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPIcs*, pages 1:1–1:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[2] Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou.
Crash-stop failures in asynchronous multiparty session types.
Logical Methods in Computer Science, 2025.

References (cont.)

[3] David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida.
Zooid: a DSL for certified multiparty computation: from mechanised metatheory to certified multiparty processes.
In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 237–251. ACM, 2021.

[4] Burak Ekici and Nobuko Yoshida.
Completeness of asynchronous session tree subtyping in Coq.
In Yves Bertot, Temur Kutsia, and Michael Norrish, editors, *15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 13:1–13:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

References (cont.)

[5] Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas, and Nobuko Yoshida.
Precise subtyping for synchronous multiparty sessions.
JLAMP, 104:127–173, 2019.

[6] Silvia Ghilezan, Jovanka Pantovic, Ivan Prokic, Alceste Scalas, and Nobuko Yoshida.
Precise Subtyping for Asynchronous Multiparty Sessions.
Proc. ACM Program. Lang., 5:16:1–16:28, jan 2021.

[7] Silvia Ghilezan, Jovanka Pantović, Ivan Prokić, Alceste Scalas, and Nobuko Yoshida.
Precise subtyping for asynchronous multiparty sessions.
ACM Trans. Comput. Logic, 24(2), Nov 2023.

[8] Jonas Kastberg Hinrichsen, Jules Jacobs, and Robbert Krebbers.
Multris: Functional verification of multiparty message passing in Separation Logic.
Proc. ACM Program. Lang., 8(OOPSLA2):1446–1474, 2024.

References (cont.)

[9] Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis.
The power of parameterization in coinductive proof.
In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 193–206. ACM, 2013.

[10] Jules Jacobs, Stephanie Balzer, and Robbert Krebbers.
Multiparty GV: functional multiparty session types with certified deadlock freedom.
Proc. ACM Program. Lang., 6(ICFP):466–495, 2022.

References (cont.)

[11] Kirstin Peters and Nobuko Yoshida.
Separation and encodability in mixed choice multiparty sessions.
In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 62:1–62:15. ACM, 2024.

[12] Dawit Legesse Tirore.
A Mechanisation of Multiparty Session Types.
PhD thesis, ITU Copenhagen, December 2024.

[13] Thien Udomsrirungruang and Nobuko Yoshida.
Top-down or bottom-up? complexity analyses of synchronous multiparty session types.
Proc. ACM Program. Lang., 9(POPL):1040–1071, 2025.

References (cont.)

[14] Yannick Zakowski, Paul He, Chung-Kil Hur, and Steve Zdancewic.
An equational theory for weak bisimulation via generalized parameterized coinduction.
In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 71–84. ACM, 2020.