Towards A Synthetic Formulation of Multiparty Session Types

David Castro-Perez, Francisco Ferreira

d.castro-perez@kent.ac.uk

10-12-2024

University of Kent

Background and Motivation

A Crash Course on Classic Multiparty Session Types

What is wrong with this code?

```go
func Worker(n int, resp chan int, err chan error) { ... }
func Master(reqCh chan int, respCh chan []int, cErrCh chan error) {
  for {
    ubound := <-reqCh
    workerChs := make([]chan int, ubound)
    errCh := make(chan error)
    for i := 0; i < ubound; i++ {
      workerChs[i] = make(chan int)
      go Worker(i+1, workerChs[i], errCh)
    }
    var res []int
    for i := 0; i < ubound; i++ {
      select {
      case sql := <-workerChs[i]:
        res = append(res, sql)
      case err := <-errCh:
        cErrCh <- err
        return
      }}
    respCh <- res}}
```

What is wrong with this code?

```
func Worker(n int, resp chan int, err chan error) { ... }
func Master(reqCh chan int, respCh chan []int, cErrCh chan error) {
  for {
    ubour
    work
    errCh
    for
      wo
      go
    }
    var
    for
      sel
      case sql := <-workerChs[i]:
        res = append(res, sql)
      case err := <-errCh:
        cErrCh <- err
        return
      }}
    respCh <- res}}
```

DEADLOCK!

ORPHAN MESSAGES!

NO RESOURCE CLEANUP!

...

What is wrong with this code?

```
func Worker(n int, resp chan int, err chan error) { ... }
func Master(reqCh chan int, respCh chan []int, cErrCh chan error) {
  for {
    ubound := <-reqCh
    worke
    errCh
    for i
      wor
      go
    }
    var res []int
    for i := 0; i < ubound; i++ {
      select {
      case sql := <-workerChs[i]:
        res = append(res, sql)
      case err := <-errCh:
        cErrCh <- err
        return
      }}
    respCh <- res}}
```

Master needs to guarantee that all Workers are notified when there is an error.

Key Idea

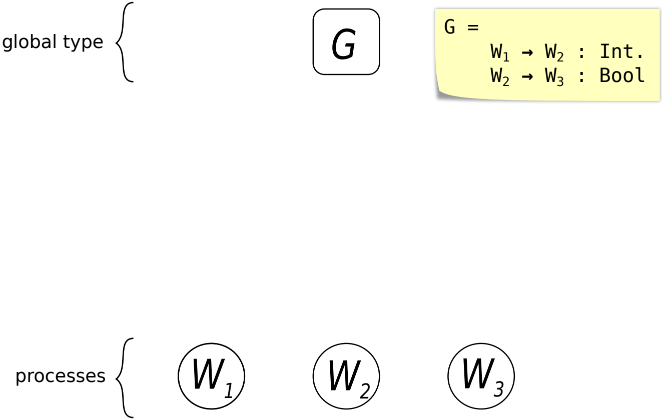
Multiparty Session Types prevent you from writing the code in the previous slide by enforcing syntactically that process implementations follow a given specification.

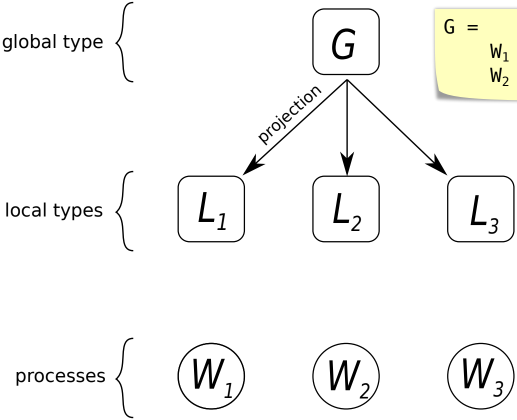
In a nutshell:

1. Global types: protocol specifications among a fixed number of different *roles*.
2. Role: sets of interactions that processes can do in a protocol.
3. Local types: protocol specifications *from the point of view of a single role*.
4. Projection: a *partial function* that extracts *local type* given a *global types* and a *role*.

5. Well-formedness: guarantees **deadlock-freedom**, usually defined in terms of *projectability*.

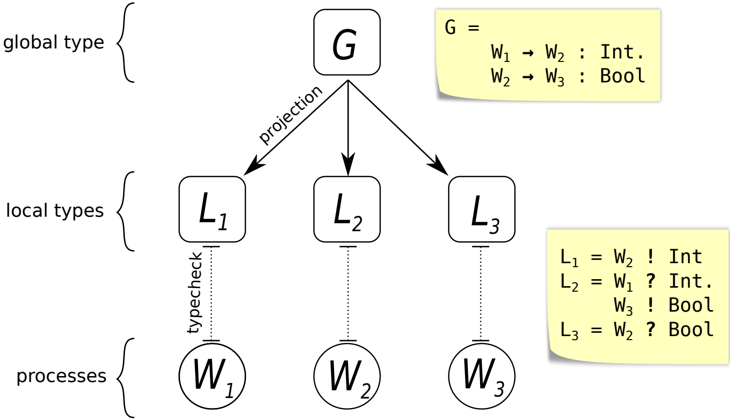

processes
W_1
W_2
W_3

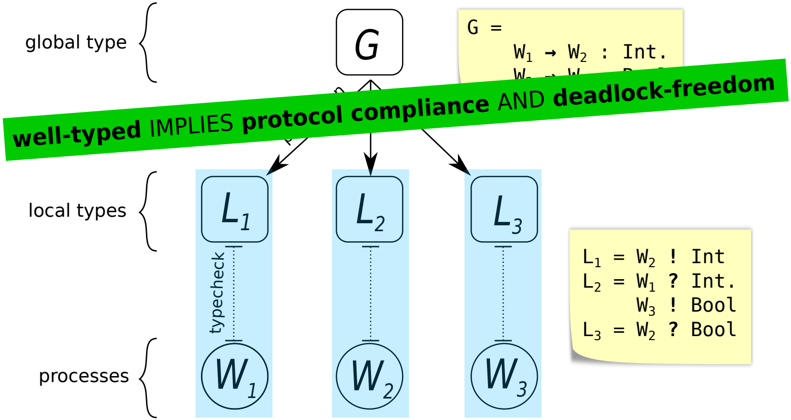

global type
G
G =
W1 → W2 : Int.
W2 → W3 : Bool
processes
W1
W2
W3



$$
\begin{aligned}
G = \quad & \\
& W_1 \rightarrow W_2 : \mathtt{Int}. \\
& W_2 \rightarrow W_3 : \mathtt{Bool}
\end{aligned}
$$

$$
\begin{aligned}
L_1 &= W_2 \,!\, \mathtt{Int} \\
L_2 &= W_1 \,?\, \mathtt{Int}. \\
&\ W_3 \,!\, \mathtt{Bool} \\
L_3 &= W_2 \,?\, \mathtt{Bool}
\end{aligned}
$$


global type
local types
processes
G
projection
L1
L2
L3
typecheck
W1
W2
W3
G =
W1 → W2 : Int.
W2 → W3 : Bool
L1 = W2 ! Int
L2 = W1 ? Int.
W3 ! Bool
L3 = W2 ? Bool


global type
local types
processes
G
G =
W1 → W2 : Int.
well-typed IMPLIES protocol compliance AND deadlock-freedom
L1
L2
L3
typecheck
W1
W2
W3
L1 = W2 ! Int
L2 = W1 ? Int.
W3 ! Bool
L3 = W2 ? Bool

Global and Local Types

Roles			$\mathsf{p}, \mathsf{q}, \ldots$	
Sorts	S	$:=$	$\texttt{bool} \mid \texttt{nat} \mid \cdots$	Basic data types.
Global Types	G	$:=$	$\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).G_i\}_{i \in I}$	Message communication.
		$\mid$	$\mu X.G$	Recursion.
		$\mid$	X	Recursion variable.
		$\mid$	$\varnothing$	End of protocol.
Local Types	L	$:=$	$\mathsf{p}!\{\ell_i(S_i).L_i\}_{i \in I}$	Send message.
		$\mid$	$\mathsf{q}?\{\ell_i(S_i).L_i\}_{i \in I}$	Receive message.
		$\mid$	$\mu X.G$	Recursion.
		$\mid$	X	Recursion variable.
		$\mid$	$\varnothing$	End of protocol.

Projection

$$
\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright \mathsf{r} = \begin{cases} \mathsf{q}!\{\ell_i(S_i).G_i \upharpoonright \mathsf{r}\}_{i\in I} & (\mathsf{r} = \mathsf{p} \land \qquad\quad \land \mathsf{p} \neq \mathsf{q}) \\ \mathsf{p}?\{\ell_i(S_i).G_i \upharpoonright \mathsf{r}\}_{i\in I} & (\qquad\quad \land \mathsf{r} = \mathsf{q} \land \mathsf{p} \neq \mathsf{q}) \\ \sqcap_{i\in I}(G_i \upharpoonright \mathsf{r}) & (\mathsf{r} \neq \mathsf{p} \land \mathsf{r} \neq \mathsf{q} \land \mathsf{p} \neq \mathsf{q}) \end{cases}
$$

$$
\mu X.G \upharpoonright \mathsf{r} = \begin{cases} \mu X.G \upharpoonright \mathsf{r} & (\mathsf{r} \in G) \\ \varnothing & (\mathsf{r} \notin G) \end{cases} \qquad X \upharpoonright \mathsf{r} = X \qquad \varnothing \upharpoonright \mathsf{r} = \varnothing
$$

Projection

$$\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).G_i\}_{i \in I} \upharpoonright \mathsf{r} = \begin{cases} \mathsf{q}!\{\ell_i(S_i).G_i \upharpoonright \mathsf{r}\}_{i \in I} & (\mathsf{r} = \mathsf{p} \wedge \qquad\qquad \wedge \mathsf{p} \neq \mathsf{q}) \\ \mathsf{p}?\{\ell_i(S_i).G_i \upharpoonright \mathsf{r}\}_{i \in I} & (\qquad\qquad \wedge \mathsf{r} = \mathsf{q} \wedge \mathsf{p} \neq \mathsf{q}) \\ \sqcap_{i \in I}(G_i \upharpoonright \mathsf{r}) & (\mathsf{r} \neq \mathsf{p} \wedge \mathsf{r} \neq \mathsf{q} \wedge \mathsf{p} \neq \mathsf{q}) \end{cases}$$

$$\mu X.G \upharpoonright \mathsf{r} = \begin{cases} \mu X.G \upharpoonright \mathsf{r} & (\mathsf{r} \in G) \\ \varnothing & (\mathsf{r} \notin G) \end{cases} \qquad X \upharpoonright \mathsf{r} = X \qquad \varnothing \upharpoonright \mathsf{r} = \varnothing$$

$$\begin{aligned} &\mathsf{p}?\{\ell_i(S_i).L_i\}_{i \in I} \sqcap \mathsf{p}?\{\ell_j(S_j).L'_j\}_{j \in J} \\ &\quad = \mathsf{p}?\{\ell_i(S_i).L_i\}_{i \in I \setminus J} \cup \{\ell_j(S_j).L'_j\}_{j \in J \setminus I} \cup \{\ell_i(S_i).L_i \sqcap L'_i\}_{i \in I \cap J} \end{aligned}$$

$$\mathsf{p}!\{\ell_i(S_i).L_i\}_{i \in I} \sqcap \mathsf{p}!\{\ell_i(S_i).L'_i\}_{i \in I} = \mathsf{p}!\{\ell_i(S_i).L_i \sqcap L'_i\}_{i \in I}$$

$$\mu X.L \sqcap \mu X.L' = \mu X.(L \sqcap L') \qquad L \sqcap L = L$$

Projection

$$
\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).G_i\}_{i\in I} \upharpoonright \mathsf{r} = \begin{cases} \mathsf{q}!\{\ell_i(S_i).G_i \upharpoonright \mathsf{r}\}_{i\in I} & (\mathsf{r} = \mathsf{p} \wedge \quad \wedge\, \mathsf{p} \neq \mathsf{q}) \\ \mathsf{p}?\{\ell_i(S_i).G_i \upharpoonright \mathsf{r}\}_{i\in I} & (\quad \wedge\, \mathsf{r} = \mathsf{q} \wedge \mathsf{p} \neq \mathsf{q}) \\ \sqcap_{i\in I}(G_i \upharpoonright \mathsf{r}) & (\mathsf{r} \neq \mathsf{p} \wedge \mathsf{r} \neq \mathsf{q} \wedge \mathsf{p} \neq \mathsf{q}) \end{cases}
$$

It gets complicated very quickly!

$$
\mu X \ldots \quad \begin{cases} \varnothing & (\mathsf{r} \notin G) \end{cases}
$$

$$
\begin{aligned}
&\mathsf{p}?\{\ell_i(S_i).L_i\}_{i\in I} \sqcap \mathsf{p}?\{\ell_j(S_j).L'_j\}_{j\in J} \\
&\quad = \mathsf{p}?\{\ell_i(S_i).L_i\}_{i\in I\setminus J} \cup \{\ell_j(S_j).L'_j\}_{j\in J\setminus I} \cup \{\ell_i(S_i).L_i \sqcap L'_i\}_{i\in I\cap J}
\end{aligned}
$$

$$
\mathsf{p}!\{\ell_i(S_i).L_i\}_{i\in I} \sqcap \mathsf{p}!\{\ell_i(S_i).L'_i\}_{i\in I} = \mathsf{p}!\{\ell_i(S_i).L_i \sqcap L'_i\}_{i\in I}
$$

$$
\mu X.L \sqcap \mu X.L' = \mu X.(L \sqcap L') \qquad L \sqcap L = L
$$

What is the point of $\sqcap$?

Consider the following protocol
– this is similar to the behaviour of the previous Go code snippet:

$$\mu X.\mathsf{p} \to \mathsf{q} : \left\{ \begin{array}{ll} \mathsf{REQ}(\texttt{nat}).\mathsf{q} \to \mathsf{r} : \mathsf{REQ}(\texttt{bool}).X \\ \mathsf{END}() \quad\quad .\mathsf{q} \to \mathsf{r} : \mathsf{END}().\mathsf{done} \end{array} \right\}$$

What is the point of $\sqcap$?

Consider the following protocol
– this is similar to the behaviour of the previous Go code snippet:

$$\mu X.\mathsf{p} \to \mathsf{q} : \left\{ \begin{array}{ll} \mathsf{REQ}(\mathtt{nat}) & .\mathsf{q} \to \mathsf{r} : \mathsf{REQ}(\mathtt{bool}).X \\ \mathsf{END}() & .\mathsf{q} \to \mathsf{r} : \mathsf{END}().\mathsf{done} \end{array} \right\}$$

Projecting r

$$\mu X.(\mathsf{q}?\mathsf{REQ}(\mathtt{bool}).X) \sqcap (\mathsf{q}?\mathsf{END}().\varnothing)$$

$$=$$

What is the point of $\sqcap$?

Consider the following protocol
– this is similar to the behaviour of the previous Go code snippet:

$$\mu X.\mathsf{p} \to \mathsf{q} : \left\{ \begin{array}{ll} \mathsf{REQ}(\mathtt{nat}).\mathsf{q} \to \mathsf{r} : \mathsf{REQ}(\mathtt{bool}).X \\ \mathsf{END}() \quad\quad .\mathsf{q} \to \mathsf{r} : \mathsf{END}().\mathsf{done} \end{array} \right\}$$

Projecting r

$$\mu X.(\mathsf{q}?\mathsf{REQ}(\mathtt{bool}).X) \sqcap (\mathsf{q}?\mathsf{END}().\varnothing)$$

$$= \mu X.\mathsf{q}?\left\{ \begin{array}{ll} \mathsf{REQ}(\mathtt{bool}).X \\ \mathsf{END}() \quad\quad .\mathsf{done} \end{array} \right\}$$

Processes and Typing

$$
\begin{array}{llcll}
\text{Process} & P & := & \mathsf{p}\,!\,\ell\langle e\rangle.P & \text{Send a message.} \\
& & \mid & \sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i & \text{Receive a message.} \\
& & \mid & \mathtt{if}\; e\; \mathtt{then}\; P\; \mathtt{else}\; P' & \text{Conditional process.} \\
& & \mid & \mathtt{rec}\; X\,.\,P & \text{Recursive process.} \\
& & \mid & X & \text{Recursion variable.} \\
& & \mid & \mathtt{done} & \text{Inactive process.}
\end{array}
$$

Process Typing (simplified)

Once we have local types, process typing is simple:

T-SEND

$$\frac{\Gamma \vdash P : L_i \qquad \Gamma \vdash e : S_i \qquad i \in I}{\Gamma \vdash \mathsf{q} \,!\, \ell_i\langle e\rangle.P : (\mathsf{p}!\{\ell_i(S_i).L_i\}_{i\in I})}$$

T-RECV

$$\frac{\Gamma, x_i : S_i \vdash P_i : L_i \qquad \forall i \in I}{\Gamma \vdash \sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i : (\mathsf{p}?\{\ell_i(S_i).L_i\}_{i\in I})}$$

Problems with Classic Formulation

1. **Too syntactic:**
   - Processes and local types must align
   - Too restrictive, rules out correct processes
   - ...
2. **Unnecessarily complex:**
   - Hard to implement/mechanise, e.g.:
     - Use of runtime coinductive global types: Our PLDI 2021 paper
     - Complex graph-based representation of MPST: Jacobs et al. (2022)
     - Graph-based reasoning and decision procedure for the equality of recursive types: Tirore et al. (2023)
   - Hard to extend
3. **Imprecise** about the uses of coinduction

Example of Imprecision in Classic MPST

"We identify $\mu X.G$ with $[\mu X.G/X]G$"

This is a common statement in proofs about MPST, which clearly specifies an equirecursive formulation, but...

1. The rules still refer to open global types with variables X
2. The rules specify when and how to unfold $\mu X.G$ – if we are using equirecursion, $\mu.$ should not be in the syntax ouf our language!

Moreover, this "identification" of a global type and its unfolding is not powerful enough. E.g.

$$\mathsf{p} \to \mathsf{q} : \mathsf{p}' \to \mathsf{q}' : G \neq \mathsf{p}' \to \mathsf{q}' : \mathsf{p} \to \mathsf{q} : G$$

This forces the use of tedious syntactic proofs about how the swapping of unrelated actions does not affect the protocol.

A Few Attempts at Simplifying the Theory

Less Is More: Multiparty Session Types Revisited

ALCESTE SCALAS, Imperial College London, UK
NOBUKO YOSHIDA, Imperial College London, UK

A Few Attempts at Simplifying the Theory

Le

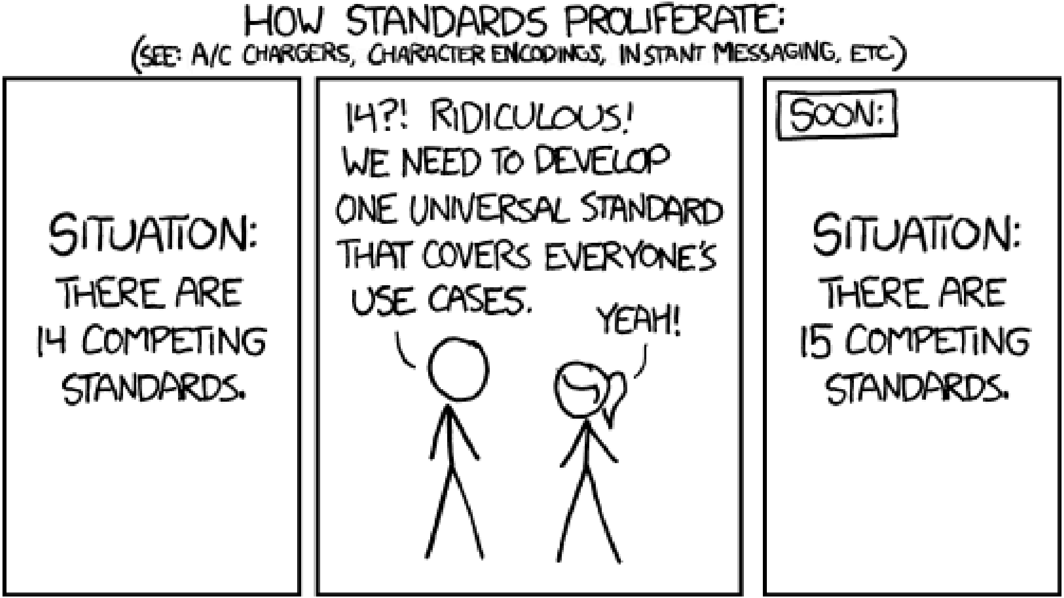
ALC
NO

Less is More Revisited

Association with Global Multiparty Session Types

Nobuko Yoshida and Ping Hou

University of Oxford, Oxford, UK
{nobuko.yoshida,ping.hou}@cs.ox.ac.uk



https://xkcd.com/927/

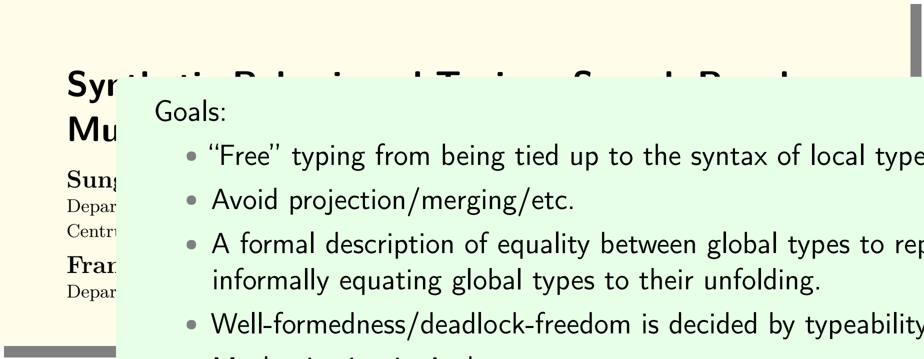
Our Approach: Synthetic Typing

Synthetic Behavioural Typing: Sound, Regular Multiparty Sessions via Implicit Local Types

Sung-Shik Jongmans ✉
Department of Computer Science, Open University, Heerlen, The Netherlands
Centrum Wiskunde & Informatica (CWI), NWO-I, Amsterdam, The Netherlands

Francisco Ferreira ✉
Department of Computer Science, Royal Holloway, University of London, UK

Our Approach: Synthetic Typing



Goals:

- "Free" typing from being tied up to the syntax of local types.
- Avoid projection/merging/etc.
- A formal description of equality between global types to replace informally equating global types to their unfolding.
- Well-formedness/deadlock-freedom is decided by typeability.
- Mechanisation in Agda.

Towards Synthetic MPST (WIP)

New (Synthetic) Core Typing Rules

New judgement : $\Gamma \vdash P : G \restriction \mathsf{p}$

T-SEND

$$\frac{\Gamma \vdash P : G' \restriction \mathsf{p} \qquad G \setminus \mathsf{p} \xrightarrow{\ell(S)} \mathsf{q} = G' \qquad \Gamma \vdash e : S}{\Gamma \vdash \mathsf{q}\,!\,\ell\langle e\rangle.P : G \restriction \mathsf{p}}$$

T-RECV

$$\frac{\Gamma, x_i : S_i \vdash P_i : G' \restriction \mathsf{p} \qquad \forall\, G \setminus \mathsf{q} \xrightarrow{\ell_i(S_i)} \mathsf{p} = G'}{\Gamma \vdash \sum_{i \in I} \mathsf{q}?\ell_i(x_i).P_i : G \restriction \mathsf{p}}$$

T-SKIP

$$\frac{\Gamma \vdash P : G' \restriction \mathsf{r} \qquad \forall\, G \setminus \alpha = G' \text{ s.t. } \mathsf{r} \notin \mathsf{parts}(\alpha)}{\Gamma \vdash P : G \restriction \mathsf{r}}$$

Synthetic, in that G' occurs only in the premise, not in the conclusion. G' needs to be *synthesised* by using the rules of the operational semantics of global types (Jongmans and Ferreira, 2023).

New

What is wrong with these rules?

T-SEND

$$\frac{\Gamma \vdash P : G' \restriction \mathsf{p} \qquad G \setminus \mathsf{p} \xrightarrow{\ell(S)} \mathsf{q} = G' \qquad \Gamma \vdash e : S}{\Gamma \vdash \mathsf{q}\,!\,\ell\langle e \rangle.P : G \restriction \mathsf{p}}$$

T-RECV

$$\frac{\Gamma, x_i : S_i \vdash P_i : G' \restriction \mathsf{p} \qquad \forall\ G \setminus \mathsf{q} \xrightarrow{\ell_i(S_i)} \mathsf{p} = G'}{\Gamma \vdash \sum_{i \in I} \mathsf{q}?\ell_i(x_i).P_i : G \restriction \mathsf{p}}$$

T-SKIP

$$\frac{\Gamma \vdash P : G' \restriction \mathsf{r} \qquad \forall\ G \setminus \alpha = G' \text{ s.t. } \mathsf{r} \notin \mathsf{parts}(\alpha)}{\Gamma \vdash P : G \restriction \mathsf{r}}$$

New

Hint: the problem is in these rules

T-RECV

$$\frac{\Gamma, x_i : S_i \vdash P_i : G' \restriction \mathsf{p} \qquad \forall\, G \setminus \mathsf{q} \xrightarrow{\ell_i(S_i)} \mathsf{p} = G'}{\Gamma \vdash \sum_{i \in I} \mathsf{q}?\ell_i(x_i).P_i : G \restriction \mathsf{p}}$$

T-SKIP

$$\frac{\Gamma \vdash P : G' \restriction \mathsf{r} \qquad \forall\, G \setminus \alpha = G' \text{ s.t. } \mathsf{r} \notin \mathsf{parts}(\alpha)}{\Gamma \vdash P : G \restriction \mathsf{r}}$$

New

Hint 2: the problem is the same in both rules, let's focus on this one

T-RECV

$$\frac{\Gamma, x_i : S_i \vdash P_i : G' \upharpoonright \mathsf{p} \qquad \forall\ G \setminus \mathsf{q} \xrightarrow{\ell_i(S_i)} \mathsf{p} = G'}{\Gamma \vdash \sum_{i \in I} \mathsf{q}?\ell_i(x_i).P_i : G \upharpoonright \mathsf{p}}$$