

# Towards A Synthetic Formulation of Multiparty Session Types

David Castro-Perez, Francisco Ferreira

[d.castro-perez@kent.ac.uk](mailto:d.castro-perez@kent.ac.uk)

10-12-2024



# Background and Motivation

A Crash Course on Classic Multiparty Session Types

# 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
        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 {
        about
        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 *projectibility*.

processes {  $W_1$     $W_2$     $W_3$  }

global type {

$G$

$G =$   
 $W_1 \rightarrow W_2 : \text{Int.}$   
 $W_2 \rightarrow W_3 : \text{Bool}$

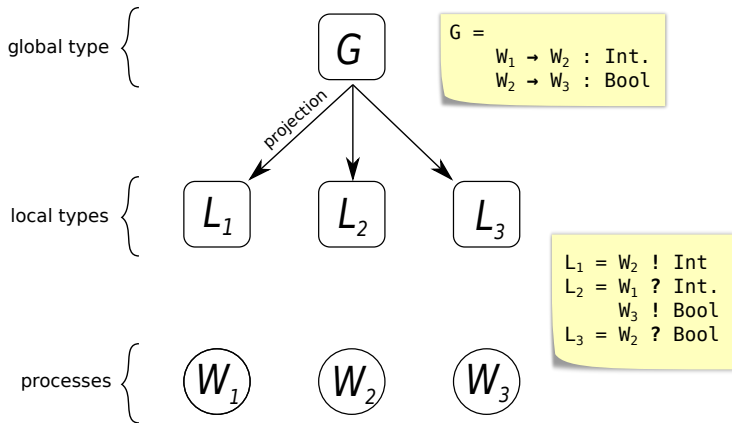
processes {

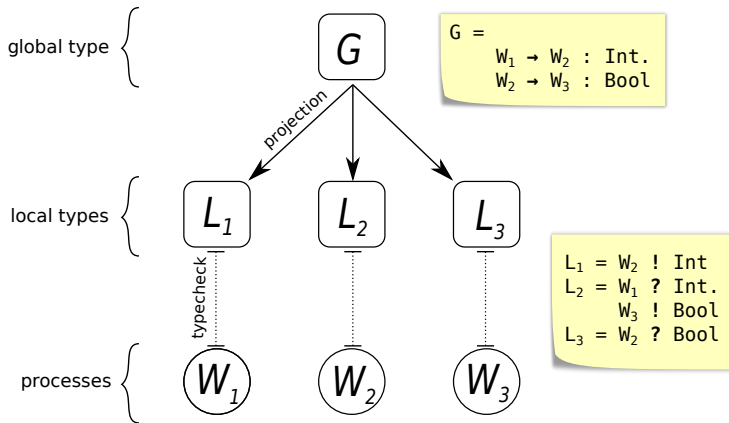
$W_1$

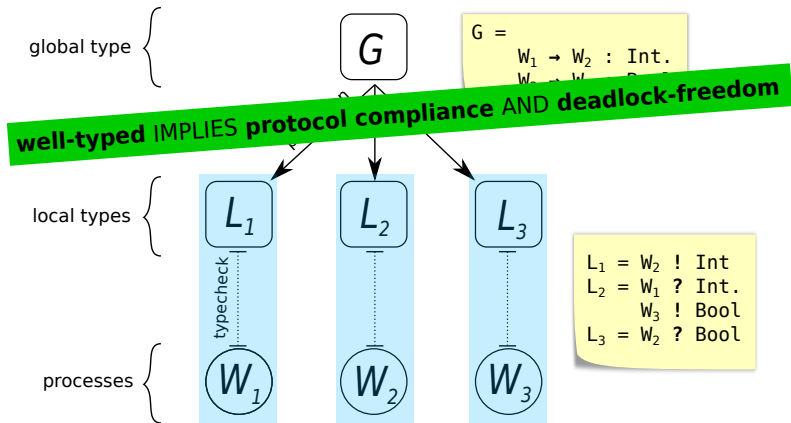
$W_2$

$W_3$









# Global and Local Types

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

# Projection

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

$$\mu X.G \upharpoonright r = \begin{cases} \mu X.G \upharpoonright r & (r \in G) \\ \emptyset & (r \notin G) \end{cases} \quad X \upharpoonright r = X \quad \emptyset \upharpoonright r = \emptyset$$

---

# Projection

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

$$\mu X.G \upharpoonright r = \begin{cases} \mu X.G \upharpoonright r & (r \in G) \\ \emptyset & (r \notin G) \end{cases} \quad X \upharpoonright r = X \quad \emptyset \upharpoonright r = \emptyset$$


---

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

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

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

# Projection

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

It gets complicated very quickly!

$$\mu \Delta. G \upharpoonright r = \begin{cases} \emptyset & (r \notin G) \end{cases} \quad \Delta \upharpoonright r = \Delta \quad \emptyset \upharpoonright r = \emptyset$$

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

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

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

# What is the point of $\sqcap$ ?

Example:

$$\mu X. p \rightarrow q : \left\{ \begin{array}{l} \text{REQ}(\text{nat}).q \rightarrow r : \text{REQ}(\text{bool}).X \\ \text{END}() \quad .q \rightarrow r : \text{END}().\text{done} \end{array} \right\}$$



# What is the point of $\sqcap$ ?

Example:

$$\mu X. p \rightarrow q : \left\{ \begin{array}{l} \text{REQ}(\text{nat}).q \rightarrow r : \text{REQ}(\text{bool}).X \\ \text{END}() \quad .q \rightarrow r : \text{END}().\text{done} \end{array} \right\}$$

Projecting  $r$

$$\mu X. (q? \text{REQ}(\text{bool}).X) \sqcap (q? \text{END}().\emptyset)$$

=

# What is the point of $\sqcap$ ?

Example:

$$\mu X. p \rightarrow q : \left\{ \begin{array}{l} \text{REQ}(\text{nat}).q \rightarrow r : \text{REQ}(\text{bool}).X \\ \text{END}() \quad .q \rightarrow r : \text{END}().\text{done} \end{array} \right\}$$

Projecting  $r$

$$\begin{aligned} & \mu X. (q? \text{REQ}(\text{bool}).X) \sqcap (q? \text{END}().\emptyset) \\ &= \mu X. q? \left\{ \begin{array}{l} \text{REQ}(\text{bool}).X \\ \text{END}() \quad .\text{done} \end{array} \right\} \end{aligned}$$

# Processes and Typing

Process	$P$	$:=$	$p!l\langle e \rangle.P$	Send a message.
			$\sum_{i \in I} p?l_i(x_i).P_i$	Receive a message.
			$\text{if } e \text{ then } P \text{ else } P'$	Conditional process.
			$\text{rec } X.P$	Recursive process.
			$X$	Recursion variable.
			$\text{done}$	Inactive process.

# Process Typing (simplified)

Once we have local types, process typing is simple:

T-SEND

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

T-RECV

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

# 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

# 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`

# HOW STANDARDS PROLIFERATE:

(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:  
THERE ARE  
14 COMPETING  
STANDARDS.

14?! RIDICULOUS!  
WE NEED TO DEVELOP  
ONE UNIVERSAL STANDARD  
THAT COVERS EVERYONE'S  
USE CASES.



SOON:

SITUATION:  
THERE ARE  
15 COMPETING  
STANDARDS.



# 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

Synthetic Typing: A New Paradigm in Type Theory

Mu

Sun

Depar

Centr

Frar

Depar

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 :  $\Gamma \vdash P : G \upharpoonright p$

T-SEND

$$\frac{G \setminus p \xrightarrow{\ell(S)} q = G' \quad \Gamma \vdash P : G' \upharpoonright p \quad \Gamma \vdash e : S}{\Gamma \vdash q ! \ell(e).P : G \upharpoonright p}$$

T-RECV

$$\frac{\forall (i \in I) \text{ s.t. } G \setminus q \xrightarrow{\ell_i(S_i)} p = G' \text{ we have } \Gamma, x_i : S_i \vdash P_i : G' \upharpoonright p}{\Gamma \vdash \sum_{i \in I} q ? \ell_i(x_i).P_i : G \upharpoonright p}$$

T-SKIP

$$\frac{\forall (i \in I) \text{ s.t. } G \setminus q \xrightarrow{\ell_i(S_i)} p = G' \text{ and } p \neq r \wedge q \neq r \text{ we have } \Gamma \vdash P : G' \upharpoonright r}{\Gamma \vdash P : G \upharpoonright r}$$

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.

# New (Synthetic) Core Typing Rules

What is wrong with these rules?

T-SEND

$$\frac{G \setminus \overset{\ell(S)}{p \rightarrow q} = G' \quad \Gamma \vdash P : G' \upharpoonright p \quad \Gamma \vdash e : S}{\Gamma \vdash q ! \ell\langle e \rangle . P : G \upharpoonright p}$$

T-RECV

$$\frac{\forall (i \in I) \text{ s.t. } G \setminus \overset{\ell_i(S_i)}{q \rightarrow p} = G' \text{ we have } \Gamma, x_i : S_i \vdash P_i : G' \upharpoonright p}{\Gamma \vdash \sum_{i \in I} q ? \ell_i(x_i) . P_i : G \upharpoonright p}$$

T-SKIP

$$\frac{\forall (i \in I) \text{ s.t. } G \setminus \overset{\ell_i(S_i)}{q \rightarrow p} = G' \text{ and } p \neq r \wedge q \neq r \text{ we have } \Gamma \vdash P : G' \upharpoonright r}{\Gamma \vdash P : G \upharpoonright r}$$

# New (Synthetic) Core Typing Rules

Hint: the problem is in these rules

T-RECV

$$\frac{\forall(i \in I) \text{ s.t. } G \setminus \overset{\ell_i(S_i)}{q \rightarrow p} = G' \text{ we have } \Gamma, x_i : S_i \vdash P_i : G' \upharpoonright p}{\Gamma \vdash \sum_{i \in I} q ? \ell_i(x_i). P_i : G \upharpoonright p}$$

T-SKIP

$$\frac{\forall(i \in I) \text{ s.t. } G \setminus \overset{\ell_i(S_i)}{q \rightarrow p} = G' \text{ and } p \neq r \wedge q \neq r \text{ we have } \Gamma \vdash P : G' \upharpoonright r}{\Gamma \vdash P : G \upharpoonright r}$$

## New (Synthetic) Core Typing Rules

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

T-RECV

$$\frac{\forall (i \in I) \text{ s.t. } G \setminus \frac{\ell_i(S_i)}{q \rightarrow p} = G' \text{ we have } \Gamma, x_i : S_i \vdash P_i : G' \upharpoonright p}{\Gamma \vdash \sum_{i \in I} q ? \ell_i(x_i). P_i : G \upharpoonright p}$$

## New (Synthetic) Core Typing Rules

What happens if  $G$  does not allow  $p$  to receive from  $q$ ?

T-RECV

$$\frac{\boxed{\forall (i \in I) \text{ s.t. } G \setminus \frac{\ell_i(S_i)}{q \rightarrow p} = G'} \quad \text{we have } \Gamma, x_i : S_i \vdash P_i : G' \upharpoonright p}{\Gamma \vdash \sum_{i \in I} q ? \ell_i(x_i). P_i : G \upharpoonright p}$$



## New (Synthetic) Core Typing Rules

This was a “rookie” mistake ... We cannot allow rules to be vacuously true!

$$\text{T-RECV} \quad \frac{\text{we have } \Gamma, x_i : S_i \vdash P_i : G' \upharpoonright p}{\Gamma \vdash \sum_{i \in I} q? \ell_i(x_i). P_i : G \upharpoonright p}$$

# (Hopefully) Fixed Typing Rules

Let  $\mathcal{R}(\mathbf{p}, \mathbf{q}, \{\ell_i(S_i)\}_{i \in I}, G) = \exists(i \in I) G', G \setminus \overset{\ell_i(S_i)}{\mathbf{p} \rightarrow \mathbf{q}} = G'$  – this means that an interaction between  $\mathbf{p}$  and  $\mathbf{q}$  is “ready” (i.e. can happen) in  $G$ .

T-SEND

$$\frac{G \setminus \overset{\ell(S)}{\mathbf{p} \rightarrow \mathbf{q}} = G' \quad \Gamma \vdash P : G' \upharpoonright \mathbf{p} \quad \Gamma \vdash e : S}{\Gamma \vdash \mathbf{q} ! \ell \langle e \rangle . P : G \upharpoonright \mathbf{p}}$$

# (Hopefully) Fixed Typing Rules

Let  $\mathcal{R}(\mathbf{p}, \mathbf{q}, \{\ell_i(S_i)\}_{i \in I}, G) = \exists(i \in I) G', G \setminus \mathbf{p} \xrightarrow{\ell_i(S_i)} \mathbf{q} = G'$  – this means that an interaction between  $\mathbf{p}$  and  $\mathbf{q}$  is “ready” (i.e. can happen) in  $G$ .

T-RECV

$$\frac{\mathcal{R}(\mathbf{q}, \mathbf{p}, \{\ell_i(S_i)\}_{i \in I}, G) \quad \forall(i \in I) \text{ s.t. } G \setminus \mathbf{q} \xrightarrow{\ell_i(S_i)} \mathbf{p} = G' \text{ we have } \Gamma, x_i : S_i \vdash P_i : G' \upharpoonright \mathbf{p}}{\Gamma \vdash \sum_{i \in I} \mathbf{q} ? \ell_i(x_i). P_i : G \upharpoonright \mathbf{p}}$$

# (Hopefully) Fixed Typing Rules

Let  $\mathcal{R}(\mathbf{p}, \mathbf{q}, \{\ell_i(S_i)\}_{i \in I}, G) = \exists(i \in I) G', G \setminus \overset{\ell_i(S_i)}{\mathbf{p} \rightarrow \mathbf{q}} = G'$  – this means that an interaction between  $\mathbf{p}$  and  $\mathbf{q}$  is “ready” (i.e. can happen) in  $G$ .

T-SKIP

$$\frac{\begin{array}{l} \mathcal{R}(\mathbf{q}, \mathbf{p}, \{\ell_i(S_i)\}_{i \in I}, G) \wedge (\mathbf{r} \notin \{\mathbf{p}, \mathbf{q}\}) \\ \forall(i \in I) \text{ s.t. } G \setminus \overset{\ell_i(S_i)}{\mathbf{q} \rightarrow \mathbf{p}} = G' \text{ we have } \Gamma \vdash P : G' \upharpoonright \mathbf{r} \end{array}}{\Gamma \vdash P : G \upharpoonright \mathbf{r}}$$

# (Hopefully) Fixed Typing Rules

Let  $\mathcal{R}(\mathbf{p}, \mathbf{q}, \{\ell_i(S_i)\}_{i \in I}, G) = \exists(i \in I) G', G \setminus \mathbf{p} \xrightarrow{\ell_i(S_i)} \mathbf{q} = G'$  – this means that an interaction between  $\mathbf{p}$  and  $\mathbf{q}$  is “ready” (i.e. can happen) in  $G$ .

T-SEND

$$\frac{G \setminus \mathbf{p} \xrightarrow{\ell(S)} \mathbf{q} = G' \quad \Gamma \vdash P : G' \upharpoonright \mathbf{p} \quad \Gamma \vdash e : S}{\Gamma \vdash \mathbf{q} ! \ell\langle e \rangle . P : G \upharpoonright \mathbf{p}}$$

T-RECV

$$\frac{\mathcal{R}(\mathbf{q}, \mathbf{p}, \{\ell_i(S_i)\}_{i \in I}, G) \quad \forall(i \in I) \text{ s.t. } G \setminus \mathbf{q} \xrightarrow{\ell_i(S_i)} \mathbf{p} = G' \text{ we have } \Gamma, x_i : S_i \vdash P_i : G' \upharpoonright \mathbf{p}}{\Gamma \vdash \sum_{i \in I} \mathbf{q} ? \ell_i(x_i) . P_i : G \upharpoonright \mathbf{p}}$$

T-SKIP

$$\frac{\mathcal{R}(\mathbf{q}, \mathbf{p}, \{\ell_i(S_i)\}_{i \in I}, G) \wedge (r \notin \{\mathbf{p}, \mathbf{q}\}) \quad \forall(i \in I) \text{ s.t. } G \setminus \mathbf{q} \xrightarrow{\ell_i(S_i)} \mathbf{p} = G' \text{ we have } \Gamma \vdash P : G' \upharpoonright r}{\Gamma \vdash P : G \upharpoonright r}$$

# (Hopefully) Fixed Typing Rules

Let  $\mathcal{R}(\mathbf{p}, \mathbf{q}, \{\ell_i(S_i)\}_{i \in I}, G) = \exists(i \in I) G', G \setminus \mathbf{p} \xrightarrow{\ell_i(S_i)} \mathbf{q} = G'$  – this means that an interaction between  $\mathbf{p}$  and  $\mathbf{q}$  is “ready” (i.e. can happen) in  $G$ .

- The rules look more complex than with a syntactic approach, but computing  $G \setminus \mathbf{q} \xrightarrow{\ell_i(S_i)} \mathbf{p} = G'$  is entirely mechanical by using the semantics of global types.
- The proof of subject reduction is greatly simplified (more in a few slides) with this formulation.
- No need of projection/merging.

T-RECV

$\mathcal{R}(\mathbf{q}, \mathbf{p}, \{\ell_i(S_i)\}_{i \in I}, G)$

$\mathbf{p}$

T-SKIP

$$\frac{\mathcal{R}(\mathbf{q}, \mathbf{p}, \{\ell_i(S_i)\}_{i \in I}, G) \wedge (r \notin \{\mathbf{p}, \mathbf{q}\}) \quad \forall(i \in I) \text{ s.t. } G \setminus \mathbf{q} \xrightarrow{\ell_i(S_i)} \mathbf{p} = G' \text{ we have } \Gamma \vdash P : G' \upharpoonright r}{\Gamma \vdash P : G \upharpoonright r}$$

# Semantics

# Examples



# Global Type Bisimilarity

# Properties of Synthetic MPST

# Wrap Up

A Crash Course on Classic Multiparty Session Types

# Benefits of Synthetic Typing

TODO