

CoMPSeT – A Framework for Comparing Multiparty Session Types

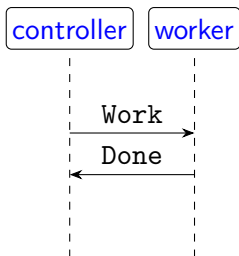
T. Ribeiro, J. Proença & M. Florido

EXPRESS/SOS @ CONFEST 2025

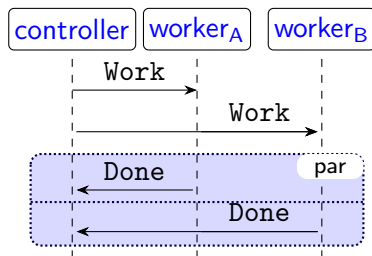
August 2025

Multiparty Session Types

It is all about ensuring communication *safety* and *liveness*



(Binary) Session Types



Multiparty Session Types

Semantical Differences

Table: Features mapping – ✓(present), ✗(absent) or N/S (not specified)

Paper	Merge criteria	Communication model	Parallel composition	Recursion scheme	Well-formedness requirements
Yoshida & Gheri	plain & full	synchronous	✗	fixed point	
Coppo et al.	plain	ordered asynchronous	✗	fixed point	
Cledou et al.	plain	ordered asynchronous	✓	✗	well-channelled
Jongmans & Proença	plain	ordered asynchronous	✓	Kleene star & fixed point	well-channelled
Guanciale & Tuosto	N/S	unordered asynchronous	N/S	N/S	N/S

Subtle semantic differences are hard to understand and compare

What We Offer

CoMPSeT - Comparing Multipart Session Types

Session

```
1 c->wA:Work ; c->wB:Work ;
2 (wA->c:Done || wB->c:Done)
```

Settings

- ☒ Semantics A
 - ☒ Merge Criteria
 - ☒ Plain
 - ☐ Full
 - ☒ Communication Model
 - ☐ Synchronous
 - ☒ Ordered Asynchronous
 - ☐ Unordered Asynchronous
 - ☒ Parallel Composition
 - ☐ Recursion Scheme
 - ☐ Fixed Point
 - ☐ Kleene Star
 - ☒ Extra Requirements
 - ☒ Well Branched
 - ☒ Well Channeled
- ☐ Semantics B
 - ☐ Merge Criteria
 - ☐ Plain
 - ☐ Full
 - ☐ Communication Model
 - ☐ Synchronous
 - ☐ Ordered Asynchronous
 - ☐ Unordered Asynchronous
 - ☐ Parallel Composition
 - ☐ Recursion Scheme
 - ☐ Fixed Point
 - ☐ Kleene Star
 - ☐ Extra Requirements
 - ☐ Well Branched
 - ☐ Well Channeled

controller-workers-v1 under the APIGenInScala3 settings

Examples

Message Sequence Chart

Global

```
1 c->wA:Work ; c->wB:Work ; (wA->c:Done || wB->c:Done)
```

Semantics A: Locals

```
1 c: wA!Work ; wB!Work ; (wA?Done || wB?Done)
2 wA: c?Work ; c!Done
3 wB: c?Work ; c!Done
```

Semantics A: Local FSMs

Semantics A: Local Compositional FSM

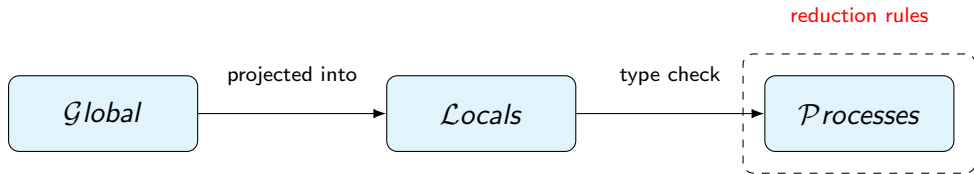
Semantics A: Iterator

Source code at CoMPSeT source.
CoMPSeT builds upon CAOS source. More concretely, our extension extended CAOS source.

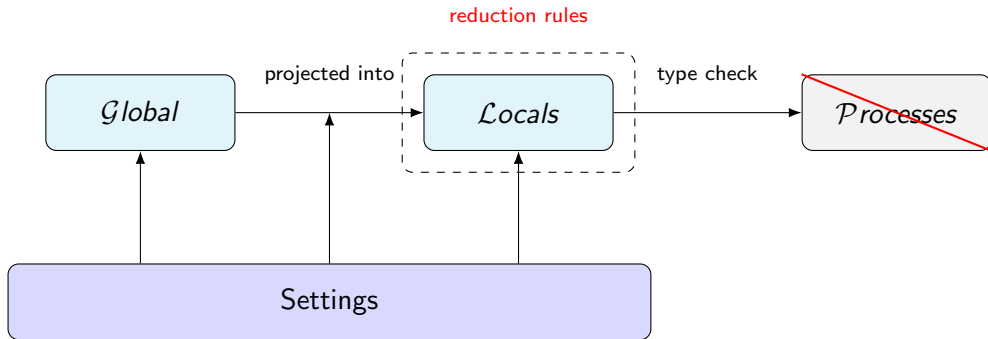
- CoMPSeT

- open source & browser-executable
- for comparing, visualising, and interacting with sessions and semantics
- Built on top of **CAOS**
 - *explained later*

Detailed Multiparty Session Types Framework



As Seen in CoMPSeT



Global and Local Types

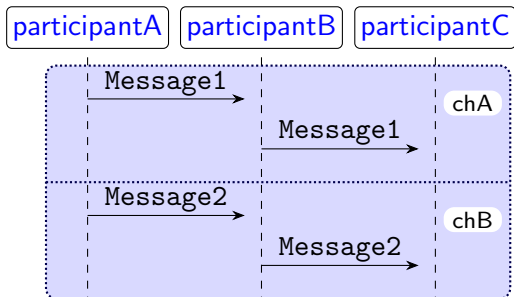
Global Type Grammar:

$$G ::= p \rightarrow q : \{t_i ; G_i\}_{1 \leq i \leq n} \mid G_1 ; G_2 \mid G_1 \parallel G_2 \mid \mu X. G \mid X \mid (G)^* \mid \text{skip}$$

Local Type Grammar:

$$L ::= pq! \{t_i ; L_i\}_{1 \leq i \leq n} \mid pq? \{t_i ; L_i\}_{1 \leq i \leq n} \mid \dots$$

Example With Branching



Global type:

```
pA → pB : {  
  m1 ; pB → pC : m1,  
  m2 ; pB → pC : m2  
}
```

Local types:

```
LpA = pB!{m1, m2}  
LpB = pA?{m1 ; pC!m1, m2 ; pC!m2}  
LpC = pB?{m1, m2}
```


Variation Points in the Semantics

While projecting

$$\begin{aligned}
 p \rightarrow q : \{t_i ; G_i\}_{1 \leq i \leq n} \downarrow_r &= pq! \{t_i ; (G_i \downarrow_r)\}_{1 \leq i \leq n} && \text{if } p = r \neq q \\
 p \rightarrow q : \{t_i ; G_i\}_{1 \leq i \leq n} \downarrow_r &= pq? \{t_i ; (G_i \downarrow_r)\}_{1 \leq i \leq n} && \text{if } p \neq r = q \\
 p \rightarrow q : \{t_i ; G_i\}_{1 \leq i \leq n} \downarrow_r &= \text{merge} (\{G_i \downarrow_r\}_{1 \leq i \leq n}) && \text{if } p \neq r \neq q \\
 &\dots
 \end{aligned}$$

While running

Assuming configurations such as $\langle M, p \rangle$, where M denotes $L_{p_1} \mid \dots \mid L_{p_n}$ and

$$\begin{aligned}
 \langle pq! \{t_i ; L_i\}_{1 \leq i \leq m} \mid M, p \cup \{pq \mapsto ts\} \rangle &\xrightarrow{pq!t_k} \langle L_k \mid M, p \cup \{pq \mapsto ts \cdot t_k\} \rangle \\
 \langle pq? \{t_i ; L_i\}_{1 \leq i \leq m} \mid M, p \cup \{pq \mapsto t_k \cdot ts\} \rangle &\xrightarrow{pq?t_k} \langle L_k \mid M, p \cup \{pq \mapsto ts\} \rangle
 \end{aligned}$$

Merge Criteria

CoMPSeT - Comparing Multiparty Session Types

Session

```
1 c->wA:Work ; c->wB:Work ;
2 (wA->c:Done || wB->c:Done)
```

Settings

- ☒ Semantics A
 - ☒ Merge Criteria
 - ☒ Plain
 - ☐ Full
 - ☒ Communication Model
 - ☐ Synchronous
 - ☒ Ordered Asynchronous
 - ☐ Unordered Asynchronous
 - ☒ Parallel Composition
 - ☐ Recursion Scheme
 - ☐ Fixed Point
 - ☐ Kleene Star
 - ☒ Extra Requirements
 - ☒ Well Branched
 - ☒ Well Channelled
- ☐ Semantics B
 - ☐ Merge Criteria
 - ☐ Plain
 - ☐ Full
 - ☐ Communication Model
 - ☐ Synchronous
 - ☐ Ordered Asynchronous
 - ☐ Unordered Asynchronous
 - ☐ Parallel Composition
 - ☐ Recursion Scheme
 - ☐ Fixed Point
 - ☐ Kleene Star
 - ☐ Extra Requirements
 - ☐ Well Branched
 - ☐ Well Channelled

controller-workers-v1 under the APIGenHScala3 settings

Examples

Message Sequence Chart

Global

Semantics A: Locals

```
1 c: wA:Work ; wB:Work ; (wA:Done || wB:Done)
2 wA: c:Work ; c:Done
3 wB: c:Work ; c:Done
```

Semantics A: Local FSMs

Semantics A: Local Compositional FSM

Semantics A: Iterator

Source code at CoMPSeT source.
CoMPSeT builds upon CAOS source. More concretely, our extension extended CAOS source.

CoMPSeT - Comparing Multiparty Session Types

Session

```
1 (pA->pB:TaskA ; pB->pC:TaskA)
2 +
3 (pA->pB:TaskB ; pB->pC:TaskB)
```

Settings

- ☒ Semantics A
 - ☒ Merge Criteria
 - ☐ Plain
 - ☒ Full
 - ☒ Communication Model
 - ☐ Synchronous
 - ☐ Ordered Asynchronous
 - ☐ Unordered Asynchronous
 - ☐ Parallel Composition
 - ☒ Recursion Scheme
 - ☒ Fixed Point
 - ☐ Kleene Star
 - ☒ Extra Requirements
 - ☒ Well Branched
 - ☐ Well Channelled
- ☒ Semantics B
 - ☒ Merge Criteria
 - ☒ Plain
 - ☐ Full
 - ☒ Communication Model
 - ☐ Synchronous
 - ☒ Ordered Asynchronous
 - ☐ Unordered Asynchronous
 - ☐ Parallel Composition
 - ☐ Recursion Scheme
 - ☒ Fixed Point
 - ☐ Kleene Star
 - ☒ Extra Requirements
 - ☒ Well Branched
 - ☐ Well Channelled

simple branching - v2 compared for both VeryGentleIntroMPST and GentleIntroMPASyncST

Examples

Message Sequence Chart

Global

Bisimulation Checker

Semantics A: Locals

```
1 pA: (pB:TaskA + pB:TaskB)
2 pB: (pA:TaskA + pC:TaskA + pA:TaskB + pC:TaskB)
3 pC: (pB:TaskA + pB:TaskB)
```

Semantics A: Local FSMs

Semantics A: Local Compositional FSM

Semantics A: Iterator

Semantics B: Locals

Semantics B: Local FSMs

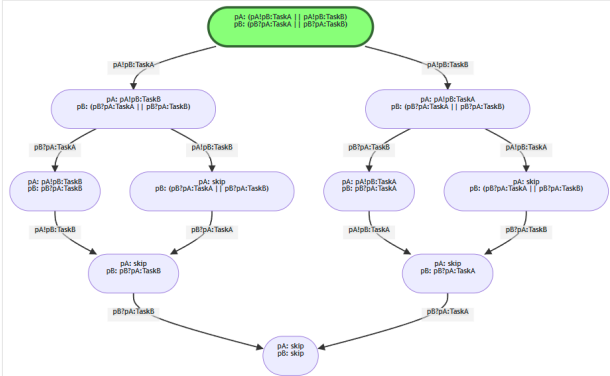
Semantics B: Local Compositional FSM

Semantics B: Iterator

Error raised by 'Semantics B: Locals':
[Plain Merge] - projection undefined for [pC] in [[pB?pA.TaskA, pB!pC.TaskA + pB?pA.TaskB, pB!pC.TaskB]]

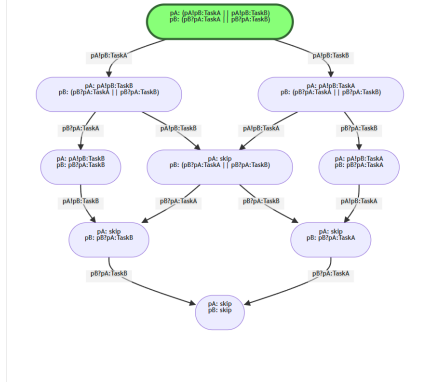
Communication Model

Semantics A: Local Compositional FSM



Ordered (Queue)

Semantics B: Local Compositional FSM



Unordered (Multiset)

For $pA \rightarrow pB : \text{TaskA} \parallel pA \rightarrow pB : \text{TaskB}$

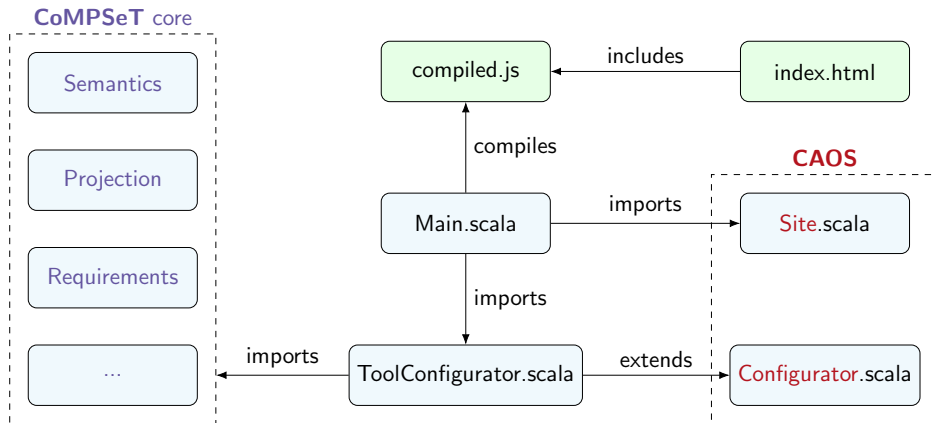
Covered Features

- Merge Criteria
- Communication Model
- Parallel Composition
- Recursion Scheme
- Extra Well-Formedness Requirements

Implementation Details

But Why CAOS?

It is all about *widgets* and ease of development



Wrap Up

What We Saw

- MPST are powerful yet fragmented
- Details may be hard to grasp
- CoMPSeT (and CAOS) enable comparisons over sessions and semantics



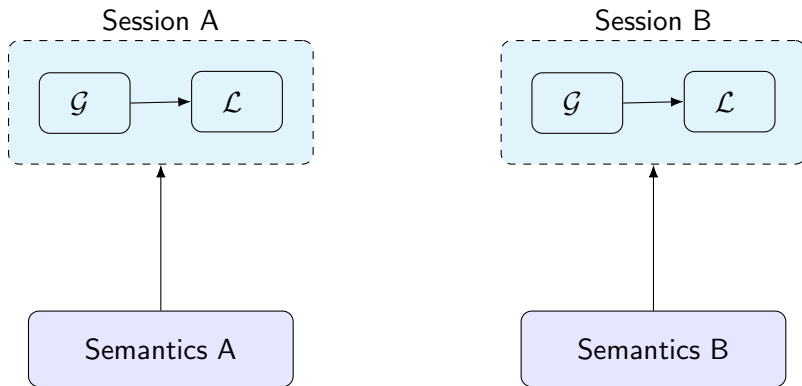
Future Work

- Additional feature assimilation
- API generation
- Formal proofs

Presented at EXPRESS/SOS @ CONFEST 2025
CoMPSeT: A Framework for Comparing Multiparty Session
Types

T. Ribeiro, J. Proença & M. Florido

Comparing Sessions and Semantics



Projection

$$skip|_r = skip$$

$$X|_r = X$$

$$(\mu X.G)|_r = \mu X.(G|_r)$$

$$(\mu X.G)|_r = skip$$

$$(G)^*|_r = (G|_r)^*$$

$$(G)^*|_r = skip$$

$$p \rightarrow q : \{t_i ; G_i\}_{1 \leq i \leq n}|_r = pq! \{t_i ; (G_i|_r)\}_{1 \leq i \leq n}$$

$$p \rightarrow q : \{t_i ; G_i\}_{1 \leq i \leq n}|_r = pq? \{t_i ; (G_i|_r)\}_{1 \leq i \leq n}$$

$$p \rightarrow q : \{t_i ; G_1\}_{1 \leq i \leq n}|_r = merge(\{G_i|_r\}_{1 \leq i \leq n})$$

$$(G_1 ; G_2)|_r = (G_1|_r) ; (G_2|_r)$$

$$(G_1 \parallel G_2)|_r = (G_1|_r) \parallel (G_2|_r)$$

undefined

if $r \in participants\{G\}$

if $r \notin participants\{G\}$

if $r \in participants\{G\}$

if $r \notin participants\{G\}$

if $p = r \neq q$

if $p \neq r = q$

if $p \neq r \neq q$

otherwise

Synchronous Semantics in CoMPSeT

$$\langle \textcolor{blue}{pq}! \{ \textcolor{green}{t}_i ; L_{1_i} \}_{1 \leq i \leq m_i} \mid \textcolor{blue}{pq}? \{ \textcolor{green}{t}_j ; L_{2_j} \}_{1 \leq j \leq m_j} \mid M \rangle \xrightarrow{\textcolor{blue}{p} \rightarrow \textcolor{blue}{q} : \textcolor{green}{t}_k} \langle L_{1_k} \mid L_{2_k} \mid M \rangle$$

Assumptions:

- The synchronous communication rule assumes no buffering mechanism, hence p is always empty and absent in the notation
- We assume that $\textcolor{green}{t}_k \in \bigcup_{i=1}^{m_i} \cap \bigcup_{j=1}^{m_j}$

Ordered Asynchronous Semantics in CoMPSeT

$$\begin{aligned} \langle \text{pq}! \{ \mathbf{t}_i ; L_i \}_{1 \leq i \leq m} \mid M, p \cup \{ \text{pq} \mapsto ts \} \rangle &\xrightarrow{\text{pq}! \mathbf{t}_k} \langle L_k \mid M, p \cup \{ \text{pq} \mapsto ts \cdot \mathbf{t}_k \} \rangle \\ \langle \text{pq} \{ \mathbf{t}_i ; L_i \}_{1 \leq i \leq m} \mid M, p \cup \{ \text{pq} \mapsto \mathbf{t}_k \cdot ts \} \rangle &\xrightarrow{\text{pq} \mathbf{t}_k} \langle L_k \mid M, p \cup \{ \text{pq} \mapsto ts \} \rangle \end{aligned}$$

Assumptions:

- a buffer $p : (\mathbb{P} \times \mathbb{P}) \rightarrow \mathbb{T}^*$, mapping each pair **sender-receiver** to a sequence of data types in \mathbb{T}

Unordered Asynchronous Semantics in CoMPSeT

$$\begin{aligned} \langle \text{pq}!\{\mathbf{t}_i; L_i\}_{1 \leq i \leq m} \mid M, p \rangle &\xrightarrow{\text{pq}!\mathbf{t}_k} \langle L_k \mid M, p \cup (\mathbf{p}, \mathbf{q}, \mathbf{t}_k) \rangle \\ \langle \text{pq}?\{\mathbf{t}_i; L_i\}_{1 \leq i \leq m} \mid M, p \cup (\mathbf{p}, \mathbf{q}, \mathbf{t}_k) \rangle &\xrightarrow{\text{pq}?\mathbf{t}_k} \langle L_k \mid M, p \rangle \end{aligned}$$

Assumptions:

- We assume $p \in \mathcal{M}(\mathbb{P} \times \mathbb{P} \times \mathbb{T})$, where $\mathcal{M}(X)$ denotes the set of all finite multisets over the set X .

Describing Widgets

```
1 lts(initialStateA, semanticsA, showStateA) /* for sync. */
2 lts(initialStateB, semanticsB, showStateB) /* for causal async. */
3 lts(initialStateC, semanticsC, showStateC) /* for non-causal async. */
```

Native CAOS

```
1 enabledCommunicationModels(path).headOption.map {
2   communicationModel =>
3     val arguments = getArguments(root, communicationModel)
4     lts(arguments.initialState, arguments.semantics, arguments.
5       showState)
6 }
```

Extended CAOS (as used in CoMPSeT)

Allowing for

- Runtime widget variability
- Setting's concept and DSL

- Concise API for