Session Subtyping and Multiparty Compatibility

using Circular Sequents

31st International Conference on Concurrency Theory (CONCUR 2020)
Adapted for Mobility Reading group 22/10/2020.

Ross Horne

Computer Science, University of Luxembourg

1-4 September 2020
Criticism 1: Deep Inference.
Criticism 1: Deep Inference.

“*I must confess we were a little bit hampered by our lack of familiarity with the calculus of structures.*”
Criticism 1: Deep Inference.

“I must confess we were a little bit hampered by our lack of familiarity with the calculus of structures.”

The calculus of structures: Developed over the past 20 years. Enables the design of analytic proof systems for non-commutative logics. It’s novelty is the use of deep inference — rules can be applied in any context.
Criticism 1: Deep Inference.

“I must confess we were a little bit hampered by our lack of familiarity with the calculus of structures.”

The calculus of structures: Developed over the past 20 years. Enables the design of analytic proof systems for non-commutative logics. It’s novelty is the use of deep inference — rules can be applied in any context.

\[ \vdash C\{ T \otimes (U \bowtie V) \} \]
\[ \vdash C\{ (T \otimes U) \bowtie V \} \]
Criticism 1: Deep Inference.

“I must confess we were a little bit hampered by our lack of familiarity with the calculus of structures.”

The calculus of structures: Developed over the past 20 years. Enables the design of analytic proof systems for non-commutative logics. It’s novelty is the use of deep inference — rules can be applied in any context.

\[ \Gamma \vdash T \otimes (U \bowtie V) \]
\[ \Gamma \vdash (T \otimes U) \bowtie V \]

The sequent calculus: The original analytic proof calculus of Gentzen. Published in 1934, so is widely understood. Rules are applied to the root connective of a formula selected from a sequence of formulae.
Criticism 1: Deep Inference.

“I must confess we were a little bit hampered by our lack of familiarity with the calculus of structures.”

The calculus of structures: Developed over the past 20 years. Enables the design of analytic proof systems for non-commutative logics. It’s novelty is the use of deep inference — rules can be applied in any context.

$\vdash C\{ T \otimes (U \supset V) \}$

$\vdash C\{ (T \otimes U) \supset V \}$

The sequent calculus: The original analytic proof calculus of Gentzen. Published in 1934, so is widely understood. Rules are applied to the root connective of a formula selected from a sequence of formulae.

$\vdash T, U, \Gamma$

$\vdash T \supset U, \Gamma$

$\vdash T, \Gamma \vdash U, \Delta$

$\vdash T \otimes U, \Gamma, \Delta$
[\text{TIMES}]
\[
T, U, \Gamma \vdash \\
T \otimes U, \Gamma \vdash
\]

[\text{PAR}]
\[
T, \Gamma_1 \vdash U, \Gamma_2 \vdash \\
T \not\otimes U, \Gamma_1, \Gamma_2 \vdash
\]

[\text{OK}]
\[
\text{OK, OK, ... OK} \vdash
\]
\[\text{[TIMES]}\]
\[
\frac{T \land U, \Gamma \vdash}{T \otimes U, \Gamma \vdash}
\]

\[\text{[PAR]}\]
\[
\frac{T, \Gamma_1 \vdash U, \Gamma_2 \vdash}{T \parallel U, \Gamma_1, \Gamma_2 \vdash}
\]

\[\text{[OK]}\]
\[
\frac{\text{OK}, \text{OK}, \ldots \text{OK} \vdash}{\text{OK}, \text{OK}, \ldots \text{OK} \vdash}
\]

\[\text{[JOIN]}\]
\[
\frac{!\lambda_j; T_j, \Gamma \vdash \text{for all } j \in I}{\bigvee_{i \in I} !\lambda_j; T_i, \Gamma \vdash}
\]

\[\text{[MEET]}\]
\[
\frac{?\lambda_j; T_j, \Gamma \vdash \text{for some } j \in I}{\bigwedge_{i \in I} ?\lambda_j; T_i, \Gamma \vdash}
\]

\[\text{[PREFIX]}\]
\[
\frac{T, U, \Gamma \vdash}{!\lambda; T, ?\lambda; U, \Gamma \vdash}
\]
[**TIMES**]
\[
\begin{array}{c}
T, U, \Gamma \vdash \\
T \otimes U, \Gamma \vdash
\end{array}
\]

[**PAR**]
\[
\begin{array}{c}
T, \Gamma_1 \vdash U, \Gamma_2 \vdash \\
T \otimes U, \Gamma_1, \Gamma_2 \vdash
\end{array}
\]

[**OK**]
\[
\begin{array}{c}
\text{OK}, \text{OK}, \ldots \text{OK} \vdash
\end{array}
\]

[**JOIN**]
\[
\begin{array}{c}
!\lambda_j; T_j, \Gamma \vdash \text{for all } j \in I \\
\bigvee_{i \in I} !\lambda_j; T_i, \Gamma \vdash
\end{array}
\]

[**MEET**]
\[
\begin{array}{c}
?\lambda_j; T_j, \Gamma \vdash \text{for some } j \in I \\
\bigwedge_{i \in I} ?\lambda_j; T_i, \Gamma \vdash
\end{array}
\]

[**PREFIX**]
\[
\begin{array}{c}
T, U, \Gamma \vdash \\
!\lambda; T, ?\lambda; U, \Gamma \vdash
\end{array}
\]

[**INTR**]
\[
\begin{array}{c}
l \subseteq J \\
T_k, U_k, \Gamma \vdash \text{for all } k \in l \\
\bigvee_{i \in I} !\lambda_i; T_i, \bigwedge_{j \in J} ?\lambda_j; U_j, \Gamma \vdash
\end{array}
\]
Criticism 2: Session type systems should feature recursion.¹

---

¹ Thanks to discussions with Mariangiola Dezani-Ciancaglini and Paola Giannini.
Criticism 2: Session type systems should feature recursion.¹

Observation 1: Most session calculi are restricted to a regular setting — a bounded number of single threaded participants.

¹Thanks to discussions with Mariangiola Dezani-Ciancaglini and Paola Giannini.
**Criticism 2:** Session type systems should feature recursion.\(^1\)

*Observation 1:* Most session calculi are restricted to a **regular** setting — a bounded number of single threaded participants.

*Observation 2:* In the regular setting, we can use **equirecursion** from type theory — fixed points are equivalent to their infinite unfoldings.

---

\(^1\) Thanks to discussions with Mariangiola Dezani-Ciancaglini and Paola Giannini.
Criticism 2: Session type systems should feature recursion.¹

Observation 1: Most session calculi are restricted to a regular setting — a bounded number of single threaded participants.

Observation 2: In the regular setting, we can use equirecursion from type theory — fixed points are equivalent to their infinite unfoldings.

Observation 3: In proof theory, such regular recursive proofs are circular proofs.

¹ Thanks to discussions with Mariangiola Dezani-Ciancaglini and Paola Giannini.
Criticism 2: Session type systems should feature recursion.¹

Observation 1: Most session calculi are restricted to a regular setting — a bounded number of single threaded participants.

Observation 2: In the regular setting, we can use equirecursion from type theory — fixed points are equivalent to their infinite unfoldings.

Observation 3: In proof theory, such regular recursive proofs are circular proofs.

Design choice: Apply an algorithmic approach to equirecursive subtyping, due to Pierce and Sangiorgi, to make proofs in the sequent calculus circular.

\[
\begin{align*}
&\text{[Fix-\(\mu\)]} \\
&\Theta \mid [\mu t. T, \Gamma] T, [\mu t. T / t], \Gamma \vdash \\
&\Theta \mid [\mu t. T], \Gamma \vdash \\
&\text{[Leaf]} \\
&\Theta \mid [\Gamma] \Gamma \vdash
\end{align*}
\]

¹Thanks to discussions with Mariangiola Dezani-Ciancaglini and Paola Giannini.
An Example

\[ \mu u. (?_1; u) \otimes \mu v. (?_2; v) \otimes \mu t. (!_1; t \lor !_2; t) \vdash [\text{TIMES}] \]

Abbreviations:

- \( \Gamma = U, V, T \)
- \( \Gamma' = U, V, !_1; T \)
- \( \Gamma'' = U, V, !_2; T \)

Multiparty Compatibility:
Proves the following threads are multiparty compatible.

\[ \mu Y. (?_1; Y) \parallel \mu Z. (?_2; Z) \parallel \mu X. (!_1; X \oplus !_2; X) \]

Subtyping:
Establishes the following subtype relation (\( U \otimes V \leq T \iff U \otimes V \otimes T \vdash \)).

\[ \mu u. (?_1; u) \otimes \mu v. (?_2; v) \leq \mu t. (!_1; t \lor !_2; t) \]

Abbreviations:
An Example

\[
\frac{U, V, T \vdash \mu u.(\lambda_1; u) \otimes \mu v.(\lambda_2; v) \otimes \mu t.(\lambda_1; t \lor \lambda_2; t)}{\text{[Fix-\(\mu\)]} \quad \text{[\text{TIMES}]}}
\]

**Abbreviations:**

\begin{align*}
U &= \mu u.(\lambda_1; u) \\
V &= \mu v.(\lambda_2; v) \\
T &= \mu t.(\lambda_1; t \lor !\lambda_2; t)
\end{align*}
An Example

Abbreviations:

\[ U = \mu u.(?\lambda_1; u) \]
\[ V = \mu v.(?\lambda_2; v) \]
\[ T = \mu t.(!\lambda_1; t \vee !\lambda_2; t) \]
\[ \Gamma = U, V, T \]
An Example

\[
\begin{align*}
&\frac{[\Gamma] U, V, !\lambda_1; T \vdash [\text{Fix-}\mu]}{[\Gamma] U, V, !\lambda_2; T \vdash [\text{Fix-}\mu]} \quad \frac{[\Gamma] U, V, !\lambda_1; T \vDash [\text{Fix-}\mu]}{[\Gamma] U, V, !\lambda_2; T \vDash [\text{Fix-}\mu]} \quad \frac{U, V, T \vdash [\text{Join}]}{\mu u.(?\lambda_1; u) \otimes \mu v.(?\lambda_2; v) \otimes \mu t.(!\lambda_1; t \vee !\lambda_2; t) \vdash [\text{Times}]}} \\
&\frac{[\Gamma] U, V, !\lambda_1; T \vdash [\text{Fix-}\mu]}{U, V, T \vdash [\text{Times}]} \\
&\frac{[\Gamma] U, V, !\lambda_1; T \vDash [\text{Fix-}\mu]}{U, V, T \vdash [\text{Join}]} \\
&\frac{U, V, T \vdash [\text{Join}]}{\mu u.(?\lambda_1; u) \otimes \mu v.(?\lambda_2; v) \otimes \mu t.(!\lambda_1; t \vee !\lambda_2; t) \vdash [\text{Times}]}
\end{align*}
\]

Abbreviations:

\[
\begin{align*}
U &= \mu u.(?\lambda_1; u) \\
V &= \mu v.(?\lambda_2; v) \\
T &= \mu t.(!\lambda_1; t \vee !\lambda_2; t) \\
\Gamma &= U, V, T
\end{align*}
\]
An Example

\[
\begin{align*}
\frac{\Gamma' \parallel \Gamma ? \lambda_1; U, V, ! \lambda_1; T \vdash \text{[PREFIX]}}{\Gamma \vdash U, V, ! \lambda_1; T \vdash} & \quad \frac{\Gamma' \parallel \Gamma ? \lambda_2; V, ! \lambda_2; T \vdash \text{[PREFIX]}}{\Gamma \vdash U, V, ! \lambda_2; T \vdash} \\
\frac{\Gamma \vdash U, V, ! \lambda_1; T \vdash \text{[PREFIX]} \quad \Gamma \vdash U, V, ! \lambda_2; T \vdash \text{[PREFIX]}}{\Gamma \vdash U, V, T \vdash \text{[JOIN]}} \\
\frac{\Gamma \vdash U, V, ! \lambda_1; T \vdash \text{[Fix-\mu]} \quad \Gamma \vdash U, V, ! \lambda_2; T \vdash \text{[Fix-\mu]}}{\mu u.(? \lambda_1; u) \otimes \mu v.(? \lambda_2; v) \otimes \mu t.(! \lambda_1; t \lor ! \lambda_2; t) \vdash \text{[TIMES]}}
\end{align*}
\]

Abbreviations:

\[
\begin{align*}
U &= \mu u.(? \lambda_1; u) \\
V &= \mu v.(? \lambda_2; v) \\
T &= \mu t.(! \lambda_1; t \lor ! \lambda_2; t) \\
\Gamma &= U, V, T \\
\Gamma' &= U, V, ! \lambda_1; T \\
\Gamma'' &= U, V, ! \lambda_2; T
\end{align*}
\]
An Example

\[
\begin{align*}
&\Gamma \vdash \text{Leaf} \quad \Gamma' \vdash \text{Leaf} \\
\frac{\Gamma, \Gamma' \vdash \text{Leaf}}{\Gamma \vdash \text{Leaf}} &\quad \frac{\Gamma' \vdash \text{Leaf}}{\Gamma \vdash \text{Leaf}} \\
\frac{\Gamma \vdash \text{Leaf}}{\Gamma, \Gamma' \vdash \text{Leaf}} &\quad \frac{\Gamma' \vdash \text{Leaf}}{\Gamma \vdash \text{Leaf}}
\end{align*}
\]

\[
\begin{align*}
&\Gamma \vdash \text{Prefix} \quad \Gamma \vdash \text{Prefix} \\
\frac{\Gamma \vdash \text{Prefix} \quad \Gamma \vdash \text{Prefix}}{\Gamma \vdash \text{Leaf}} &\quad \frac{\Gamma \vdash \text{Prefix} \quad \Gamma \vdash \text{Prefix}}{\Gamma \vdash \text{Leaf}} \\
\frac{\Gamma \vdash \text{Prefix} \quad \Gamma \vdash \text{Prefix}}{\Gamma \vdash \text{Leaf}} &\quad \frac{\Gamma \vdash \text{Prefix} \quad \Gamma \vdash \text{Prefix}}{\Gamma \vdash \text{Leaf}}
\end{align*}
\]

\[
\begin{align*}
&\Gamma \vdash \text{Fix-µ} \\
\frac{\Gamma \vdash \text{Fix-µ} \quad \Gamma \vdash \text{Fix-µ}}{\Gamma \vdash \text{Fix-µ}} &\quad \frac{\Gamma \vdash \text{Fix-µ} \quad \Gamma \vdash \text{Fix-µ}}{\Gamma \vdash \text{Fix-µ}} \\
\frac{\Gamma \vdash \text{Fix-µ} \quad \Gamma \vdash \text{Fix-µ}}{\Gamma \vdash \text{Fix-µ}} &\quad \frac{\Gamma \vdash \text{Fix-µ} \quad \Gamma \vdash \text{Fix-µ}}{\Gamma \vdash \text{Fix-µ}}
\end{align*}
\]

\[
\begin{align*}
&\Gamma \vdash \text{Join} \\
\frac{\Gamma \vdash \text{Join} \quad \Gamma \vdash \text{Join}}{\Gamma \vdash \text{Join}} &\quad \frac{\Gamma \vdash \text{Join} \quad \Gamma \vdash \text{Join}}{\Gamma \vdash \text{Join}} \\
\frac{\Gamma \vdash \text{Join} \quad \Gamma \vdash \text{Join}}{\Gamma \vdash \text{Join}} &\quad \frac{\Gamma \vdash \text{Join} \quad \Gamma \vdash \text{Join}}{\Gamma \vdash \text{Join}}
\end{align*}
\]

Abbreviations:
\[
\begin{align*}
U &= \mu u. (?\lambda_1; u) \\
V &= \mu v. (?\lambda_2; v) \\
T &= \mu t. (!\lambda_1; t \lor !\lambda_2; t) \\
\Gamma &= U, V, T \\
\Gamma' &= U, V, !\lambda_1; T \\
\Gamma'' &= U, V, !\lambda_2; T
\end{align*}
\]
An Example

\[
\begin{array}{c}
[\Gamma'] \vdash \Gamma \\
\Gamma' \vdash \Gamma \quad \text{[LEAF]} \\
[\Gamma'] \vdash \Gamma \quad \text{[PREFIX]} \\
\Gamma \quad \text{[F\textsc{i}x-}\mu]\quad \\
[\Gamma'] \vdash \Gamma \quad \text{[PREFIX]} \\
\Gamma \quad \text{[F\textsc{i}x-}\mu]\quad \\
\Gamma \quad \text{[JOIN]} \\
\Gamma \quad \text{[F\textsc{i}x-}\mu]\quad \\
\Gamma \quad \text{[T\textsc{i}mes]} \\
\end{array}
\]

Abbreviations:

\[
\begin{align*}
U &= \mu u. (\lambda 1; u) \\
V &= \mu v. (\lambda 2; v) \\
T &= \mu t. (\lambda 1; t \lor !\lambda 2; t) \\
\Gamma &= U, V, T \\
\Gamma' &= U, V, !\lambda 1; T \\
\Gamma'' &= U, V, !\lambda 2; T \\
\end{align*}
\]

Multiparty Compatibility: Proves the following threads are multiparty compatible.

\[
\mu Y. (?\lambda 1; Y) \parallel \mu Z. (?\lambda 2; Z) \parallel \mu X. (!\lambda 1; X \oplus !\lambda 2; X)
\]
An Example

Abbreviations:  \[ \begin{align*} 
U &= \mu u.(?\lambda_1;u) \\
V &= \mu v.(?\lambda_2;v) \\
T &= \mu t.(!\lambda_1;t \lor !\lambda_2;t) \\
\Gamma &= U, V, T \\
\Gamma' &= U, V, !\lambda_1;T \\
\Gamma'' &= U, V, !\lambda_2;T 
\end{align*} \]

Multiparty Compatibility: Proves the following threads are multiparty compatible.

\[ \mu Y.(?\lambda_1;Y) || \mu Z.(?\lambda_2;Z) || \mu X.(!\lambda_1;X \oplus !\lambda_2;X) \]

Subtyping: Establishes the following subtype relation (\(U \otimes V \leq T\) iff \(U \otimes V \otimes T \vdash\)).

\[ \mu u.(?\lambda_1;u) \otimes \mu v.(?\lambda_2;v) \leq \mu t.(?\lambda_1;t \land ?\lambda_2;t) \]
The Cut Elimination “Gold Mine” (again)

Theorem (cut elimination)

$\frac{\Gamma_1, T \vdash \overline{T}, \Gamma_2 \vdash}{\Gamma_1, \Gamma_2 \vdash}$ is admissible in Session.

Corollary (algorithmic subtyping)

Subtyping is a decidable preorder.

Theorem (algorithmic typing)

All instances of

$\frac{\Delta \vdash P : T \leq U}{\Delta \vdash P : U}$

can be pushed to the bottom of a type derivation.

Theorem (deadlock freedom)

Any race-free multiparty-compatible network satisfies deadlock freedom.

Corollary (substitution principle)

$P$ can replace $Q$ while preserving multiparty compatibility, whenever

$\frac{T \leq U}{\vdash P : T}$ and $\frac{\vdash Q : U}$.
The Cut Elimination “Gold Mine” (again)

Theorem (cut elimination)

\[ \text{[Cut]} \]

\[ \Gamma_1, T \vdash \overline{T}, \Gamma_2 \vdash \]

The rule \( \frac{\Gamma_1, T \vdash \overline{T}, \Gamma_2 \vdash}{\Gamma_1, \Gamma_2 \vdash} \) is admissible in Session.

Corollary (algorithmic subtyping)

Subtyping is a decidable preorder.

Theorem (algorithmic typing)

\[ \text{[Subsumption]} \]

All instances of \( \frac{\Delta \vdash P : T \quad T \leq U}{\Delta \vdash P : U} \) can be pushed to the bottom of a type derivation.

Theorem (deadlock freedom)

Any race-free multiparty-compatible network satisfies deadlock freedom.

Corollary (substitution principle)

\( P \) can replace \( Q \) while preserving multiparty compatibility, whenever \( T \leq U \), where \( \vdash P : T \) and \( \vdash Q : U \).
Now I see! So, what cool things can you do?
Owner:

?login_page(app_ID, scope);
!deny ⊕ !authorise(name, password)

Trusted App:

!login_page(app_ID, scope);
?deny;!release
+ ?authorise(name, password);
  rec Y. !release
    ⊕ !request(token);
    ?revoke + ?response(data); Y

Resource:

rec X. ?release
+ ?request(token);
!revoke ⊕ !response(data); X
Trusted App:

\[\text{!login\_page(app\_ID, scope); deny; !release}\]
\[\text{+ ?authorise(name, password); rec Y. !release}\]
\[\oplus !\text{request(token)}; ?\text{revoke} + ?\text{response(data)}; Y\]

Untrusted App:

\[\text{!initiate(add\_ID, scope); close}\]
\[\text{+ ?authorisation\_code(code); exchange(app\_ID, secret, code); close}\]
\[\text{+ ?access\_token(token); rec Y. !request(token); ?revoke}\]
\[\text{+ ?response(data); Y}\]
OAuth 2.0 Server:

\begin{align*}
? & \text{initiate(app\_ID, scope);} \\
& \text{!login\_page(app\_ID, scope);} \\
& (\text{?deny;!close;!release}) \\
& + \ ? \text{authorise(name, password);} \\
& \quad (\text{!close;!release}) \\
& \quad \oplus \ ? \text{authorisation\_code(code);} \\
& \quad ? \text{exchange(app\_ID, secret, code);} \\
& \quad (\text{!close;!release}) \\
& \quad \oplus \ ? \text{access\_token(token)}
\end{align*}

Untrusted App:

\begin{align*}
? & \text{initiate(add\_ID, scope);} \\
& \text{?close} \\
& + \ ? \text{authorisation\_code(code);} \\
& \quad ? \text{exchange(app\_ID, secret, code);} \\
& \quad \text{?close} \\
& \quad + \ ? \text{access\_token(token);} \\
& \quad \text{?revoke} \\
& \quad + \ ? \text{response(data); Y}
\end{align*}

Untrusted App $\otimes$ OAuth Server $\leq$ Trusted App
An application that delegates to an Oauth 2.0 server

![Diagram of Oauth 2.0 process]

- **Resource**
- **Untrusted App**
- **OAuth Server**
- **Owner**

**Initiate** (app ID, scope) ↔ **begin delegation**

**Login page** (app ID, scope) ↔ **authorize** (name, password)

**End delegation**

**Authorisation code** (code) ↔ **exchange** (app ID, secret, code)

**Access token** (token) ↔ **request** (token)

**Response** (data)

**Recursion**

**Revoke**

**Choice at Resource**
Allowing the deputy to make a choice is useful
Internal delegation may liberate multiparty subtyping with roles.

Trusted App:

Owner!login_page(app_ID, scope);
?deny;!release
+ Owner?authorise(name, password);
   rec Y;!release
⊕ Resource!request(token);
   ?revoke + Resource?response(data); Y

App?initiate(app_ID, scope);
App◦●
Owner!login_page(app_ID, scope);
(?deny; ●)°App;!close;!release)
+ Owner?authorise(name, password);
   ●)°App;
   (!close;!release)
⊕ App!authorisation_code(code);
   App?exchange(app_ID, secret, code);
   (!close;!release)
⊕ App!access_token(token)

OAuth!initiate(add_ID, scope);
○ ●OAuth; OAuth● ○;
?close
+ OAuth?authorisation_code(code);
   OAuth!exchange(app_ID, secret, code);
   ?close
+ OAuth?access_token(token);
   rec Y. OAuth!request(token);
   ?revoke
   + Resource?response(data); Y
Conclusion and discussion

**Conclusion:** Non-commutative logic + race-freedom provides us with rich notions of multiparty compatibility and subtyping.

**Discussion:** The follow has no global type, but is deadlock free and both "Kobayashi" and "Padovani" live (LIVE and LIVE+ respectively in POPL '19). System S session verifies this (but only guarantees deadlock freedom without further modifications).

\[
\mu X. (!\lambda_1; X \oplus !\lambda_2) \parallel \mu Y. (?\lambda_1; Y + ?\lambda_2) \parallel \mu X. (!\lambda_3; X \oplus !\lambda_4) \parallel \mu Y. (?\lambda_3; Y + ?\lambda_4)
\]

**Question for the Mobility Reading Group:** What established extensions of global types allow the above to be typed and also guarantee livelock freedom (or, at least, deadlock freedom)?
Conclusion and discussion

**Conclusion:** Non-commutative logic + race-freedom provides us with rich notions of multiparty compatibility and subtyping.

**Discussion:** The follow has no global type, but is deadlock free and both “Kobayashi” and “Padovani” live (LIVE and LIVE+ respectively in POPL’19). System Session verifies this (but only guarantees deadlock freedom without further modifications).

\[
\mu X. (\!\lambda_1; X \oplus \!\lambda_2) \parallel \mu Y. (?\lambda_1; Y + \!\lambda_2) \parallel \mu X. (\!\lambda_3; X \oplus !\lambda_4) \parallel \mu Y. (?\lambda_3; Y + ?\lambda_4)
\]
Conclusion and discussion

**Conclusion:** Non-commutative logic + race-freedom provides us with rich notions of multiparty compatibility and subtyping.

**Discussion:** The follow has no global type, but is deadlock free and both “Kobayashi” and “Padovani” live (LIVE and LIVE+ respectively in POPL’19). System Session verifies this (but only guarantees deadlock freedom without further modifications).

\[
\mu X. (\lambda_1; X \oplus !\lambda_2) \parallel \mu Y. (\lambda_1; Y + \lambda_2) \parallel \mu X. (\lambda_3; X \oplus !\lambda_4) \parallel \mu Y. (\lambda_3; Y + \lambda_4)
\]

**Question for the Mobility Reading Group:** What established extensions of global types allow the above to be typed and also guarantee livelock freedom (or, at least, deadlock freedom)?