Partially Typed Multiparty Sessions

Franco Barbanera\textsuperscript{1}, Mariangiola Dezani \textsuperscript{2}

\textsuperscript{1} University of Catania
\textsuperscript{2} University of Torino

ICE - June 19, 2023, Lisbon
OVERVIEW

▶ Choreographies and MPSTs: a type assignment approach;

▶ Lock-freedom: egalitarianism is not for system components;

▶ ICE’23: A MPST type assignment for classist Lock-freedom.
OVERVIEW

- Choreographies and MPSTs: a type assignment approach;

- Lock-freedom: egalitarianism is not for system components;

OVERVIEW

- Choreographies and MPSTs: a type assignment approach;
- Lock-freedom: egalitarism is not for system components;
OVERVIEW

- Choreographies and MPSTs: a type assignment approach;
- Lock-freedom: egalitarianism is not for system components;
MultiParty Session Types (MPST):

a body of coreographic formalisms
MultiParty Session Types (MPST):

a body of coreographic formalisms

where
two distinct but related views of a concurrent systems do coexist:
MultiParty Session Types (MPST):

a body of coreographic formalisms

where

two distinct but related views of a concurrent systems do coexist:

**global view:** overall behaviour of the system formalised using
the notion of Global Type
MultiParty Session Types (MPST): a body of coreographic formalisms

where
two distinct but related views of a concurrent systems do coexist:

**global view:** overall behaviour of the system formalised using the notion of Global Type

**local view:** behaviours of the single components in suitable process algebras
**Top-down MPST**: communication protocols are explicitly described as global types and, subsequently, by projecting them, local types are obtained for implementation.
MPST approaches

**Top-down MPST:** communication protocols are explicitly described as global types and, subsequently, by projecting them, local types are obtained for implementation.
MPST approaches

**Top-down MPST:** communication protocols are explicitly described as global types and, subsequently, by projecting them, local types are obtained for implementation.
**Top-down MPST:** communication protocols are explicitly described as global types and, subsequently, by projecting them, local types are obtained for implementation.

Systems obtained by projecting (well-formed) global types enjoy good communication properties.
MPST approaches

**Bottom-up MPST:** no projection is used and local behaviours are checked against global types by means of a type assignment system.
MPST approaches

**Bottom-up MPST:** no projection is used and local behaviours are checked against global types by means of a type assignment system.
MPST approaches

**Bottom-up MPST:** no projection is used and local behaviours are checked against global types by means of a type assignment system.
MPST approaches

**Bottom-up MPST:** no projection is used and local behaviours are checked against global types by means of a type assignment system.

Systems typable with (well-formed) global types enjoy good communication properties.
A “bottom-up” MPST
A “bottom-up” MPST

Calculus of Sessions and its type system

[B., Dezani et al. FACS’22]
A “bottom-up” MPST

Calculus of Sessions and its type system
[B., Dezani et al. FACS’22]

**Processes**

\[ P ::= \text{coind} \ 0 \mid p!\{\lambda_i.P_i\}_{i \in I} \mid p?\{\lambda_i.P_i\}_{i \in I} \]
A “bottom-up” MPST

Calculus of Sessions and its type system  
[B., Dezani et al. FACS’22]  

Processes  

\[ P ::= \frac{coind}{0} | p\!\{\lambda_i. P_i\}_{i \in I} | p?\{\lambda_i. P_i\}_{i \in I} \]

Multiparty Sessions  

\[ \mathcal{M} = p_1[P_1] \parallel \cdots \parallel p_n[P_n] \]
A “bottom-up” MPST

Calculus of Sessions and its type system
[B., Dezani et al. FACS’22]

Processes

\[ P ::= \text{coind} \ 0 \mid p!\{\lambda_P\}_i \mid p?\{\lambda_P\}_i \]

Multiparty Sessions

\[ \mathbb{M} = p_1[P_1] \parallel \cdots \parallel p_n[P_n] \]

(synchronous) Operational Semantics

\[
\ell \in I \subseteq J
\]

\[ p[q!\{\lambda_P\}_i] \parallel q[p?\{\lambda_P\}_j] \parallel \mathbb{M} \xrightarrow{p^\lambda q} p[P_\ell] \parallel q[Q_\ell] \parallel \mathbb{M} \]
A session example: carol, sam and tom

c[s!\{ok.t?ok, ko\}] || s[c?\{ok.t!ok, ko.t!ko\}] || t[s?\{ok.c!ok, ko\}]

A session example: carol, sam and tom

\[ \text{c}[s!\{\text{ok}.t?\text{ok}, \text{ko}\}] || s[c?\{\text{ok}.t!\text{ok}, \text{ko}.t!\text{ko}\}] || t[s?\{\text{ok}.c!\text{ok}, \text{ko}\}] \]
A session example: carol, sam and tom

\[
c[s\{\text{ok.t?ok, ko}\}] \parallel s[c\{\text{ok.t!ok, ko.t!ko}\}] \parallel t[s\{\text{ok.c!ok, ko}\}]
\]
A session example: carol, sam and tom

\[ c[s!\{ok.t?ok, ko\}] \parallel s[c?\{ok.t!ok, ko.t!ko\}] \parallel t[s?\{ok.c!ok, ko\}] \]
A session example: carol, sam and tom

\[
c[s!\{\text{OK}.t?\text{OK}, \text{KO}\}] \parallel s[c?\{\text{OK}.t!\text{OK}, \text{KO}.t!\text{KO}\}] \parallel t[s?\{\text{OK}.c!\text{OK}, \text{KO}\}]
\]
A session example: carol, sam and tom

Reduction example

\[
c[s\{\text{ok}\cdot t?\text{ok},\text{ko}\}] \parallel s[c\{\text{ok}\cdot t!\text{ok},\text{ko}\cdot t!\text{ko}\}] \parallel t[s\{\text{ok}\cdot c!\text{ok},\text{ko}\}]
\]
A session example: carol, sam and tom

Reduction example

\[ c[s\{\text{OK}.t?\text{OK},\text{KO}\}] \parallel s[c?\{\text{OK}.t!\text{OK},\text{KO}.t!\text{KO}\}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}] \]

\[ \Downarrow \]

\[ c[t?\text{OK}] \parallel s[t!\text{OK}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}] \]
A session example: carol, sam and tom

Reduction example

\[
c[s\{\text{OK}.t?\text{OK},\text{KO}\}] \parallel s[c?\{\text{OK}.t!\text{OK},\text{KO}.t!\text{KO}\}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}]
\]

\[
c[s\{\text{OK}.t?\text{OK},\text{KO}\}] \parallel s[c?\{\text{OK}.t!\text{OK},\text{KO}.t!\text{KO}\}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}]
\]

\[
c[t?\text{OK}] \parallel s[t!\text{OK}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}]
\]

\[
c[t?\text{OK}] \parallel s[0] \parallel t[c!\text{OK}]
\]
A session example: carol, sam and tom

Reduction example

\[
\begin{align*}
  c[s\{\text{OK}.t?\text{OK},\text{KO}\}] & \parallel s[c\{\text{OK}.t!\text{OK},\text{KO}.t!\text{KO}\}] & \parallel t[s\{\text{OK}.c!\text{OK},\text{KO}\}] \\
  & \parallel \downarrow \text{COKS} \\
  & \parallel \downarrow \text{sOKT} \\
  & \parallel \downarrow \text{tOKC} \\
  & \downarrow \text{c[0]} \parallel s[0] \parallel t[0]
\end{align*}
\]
A session example: carol, sam and tom

Reduction example

\[ c[s!\{\text{ok.t?ok,ko}\}] \parallel s[c?\{\text{ok.t!ok,ko.t!ko}\}] \parallel t[s?\{\text{ok.c!ok,ko}\}] \]
A session example: carol, sam and tom

Reduction example

\[
\begin{align*}
& c[s!\{\text{OK}.t?\text{OK},\text{KO}\}] \parallel s[c?\{\text{OK}.t!\text{OK},\text{KO}.t!\text{KO}\}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}] \\
& \quad \mid \text{CKOS} \\
& \downarrow \\
& c[0] \parallel s[t!\text{KO}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}]
\end{align*}
\]
A session example: carol, sam and tom

Reduction example

\[
\begin{align*}
&c[s\{\text{OK}.t?\text{OK},\text{KO}\}] \parallel s[c?\{\text{OK}.t!\text{OK},\text{KO}.t!\text{KO}\}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}] \\
&\quad \mid \\
&\quad \quad \text{CKOS} \\
&\quad \downarrow \\
&\quad c[0] \parallel s[t!\text{KO}] \parallel t[s?\{\text{OK}.c!\text{OK},\text{KO}\}] \\
&\quad \mid \\
&\quad \quad \text{SKOT} \\
&\quad \downarrow \\
&\quad c[0] \parallel s[0] \parallel t[0]
\end{align*}
\]
Type system for the session calculus
Type system for the session calculus

**Global Types**

\[ G ::= \text{coind} \quad \text{End} \quad | \quad p \rightarrow q : \{ \lambda i . G_i \}_{i \in I} \]
Global Types

\[ G ::= \text{coind} \quad \text{End} \mid p \to q : \{ \lambda_i G_i \}_{i \in I} \]

A global type for the example

\[ c \to s : \{ \text{OK} . s \to t : \text{OK} . t \to c : \text{OK}, \text{KO} . s \to t : \text{KO} \} \]
Type system for the session calculus

Global Types

\[ G ::= \text{coind} \mid \text{End} \mid p \to q : \{ \lambda_i. G_i \}_{i \in I} \]

Typing Rules
Type system for the session calculus

Global Types

\[ G ::= \text{coind} \quad \text{End} \mid p \to q : \{ \lambda_i.G_i \}_{i \in I} \]

Typing Rules

\[ \text{End} \vdash p[0] \]
Type system for the session calculus

Global Types

\[ G ::= \text{coind} \quad \text{End} \mid p \to q : \{\lambda_i. G_i\}_{i \in I} \]

Typing Rules

\[ \text{End} \vdash p[0] \]

\[ G_i \vdash p[P_i] \parallel q[Q_i] \parallel M \quad \text{prt}(G_i) \setminus \{p, q\} = \text{prt}(M) \quad \forall i \in I \]

\[ p \to q : \{\lambda_i. G_i\}_{i \in I} \vdash p[q!\{\lambda_i. P_i\}_{i \in I}] \parallel q[p?\{\lambda_j. Q_j\}_{j \in J}] \parallel M \quad I \subseteq J \]
Type system for the session calculus

Example of type derivation

\[ \text{End} \vdash c[0] \parallel s[0] \parallel t[0] \]
Type system for the session calculus

Example of type derivation

\[ \text{End} \vdash c[0] \parallel s[0] \parallel t[0] \]

\[ t \rightarrow c: \text{OK} \vdash c[t? \text{OK}] \parallel s[0] \parallel t[c! \text{OK}] \]
Type system for the session calculus

Example of type derivation

\[
\begin{align*}
\text{End} & \vdash c[0] \parallel s[0] \parallel t[0] \\
\quad & \vdash t \rightarrow c: \text{ok} \quad \vdash c[t?\text{ok}] \parallel s[0] \parallel t[c!\text{ok}] \\
\quad & \vdash s \rightarrow t: \text{ok} \quad t \rightarrow c: \text{ok} \quad \vdash c[t?\text{ok}] \parallel s[t!\text{ok}] \parallel t[s\{\text{ok}.c!\text{ok},\text{ko}\}]
\end{align*}
\]
Type system for the session calculus

Example of type derivation

\[ \text{End} \vdash c[0] \parallel s[0] \parallel t[0] \]

\[ \begin{array}{c}
\text{t} \rightarrow c:ok \vdash c[t?ok] \parallel s[0] \parallel t[c!ok] \\
\text{s} \rightarrow t:ok \rightarrow c:ok \vdash c[t?ok] \parallel s[t!ok] \parallel t[s\{\text{ok.c!ok,ko}\}] \\
\end{array} \]

\[ \begin{array}{c}
\text{End} \vdash c[0] \parallel s[0] \parallel t[0] \\
\text{s} \rightarrow t:ko \vdash c[0] \parallel s[t!ko] \parallel t[s\{\text{ok.c!ok,ko}\}] \\
\end{array} \]
Type system for the session calculus

Example of type derivation

\[ \text{End } \vdash c[0] \parallel s[0] \parallel t[0] \]

\[ \text{t } \rightarrow c:ok \vdash c[t?ok] \parallel s[0] \parallel t[c!ok] \]

\[ \text{End } \vdash c[0] \parallel s[0] \parallel t[0] \]

\[ \text{s } \rightarrow t:ok.t \rightarrow c:ok \vdash c[t?ok] \parallel s[t!ok] \parallel t[s?\{ok.c!ok,ko\}] \]

\[ \text{c } \vdash s : \{ok.s \rightarrow pt : ok.t \rightarrow c : ok, ko.s \rightarrow t : ko\} \vdash c[s!\{ok.t?ok,ko\}] \parallel s[c?\{ok.t!ok,ko.t!ko\}] \parallel t[s?\{ok.c!ok,ko\}] \]

\[ \text{s } \rightarrow t:ko \vdash c[0] \parallel s[t!ko] \parallel t[s?\{ok.c!ok,ko\}] \]

\[ \text{c } \vdash s : \{ok.s \rightarrow pt : ok.t \rightarrow c : ok, ko.s \rightarrow t : ko\} \vdash c[s!\{ok.t?ok,ko\}] \parallel s[c?\{ok.t!ok,ko.t!ko\}] \parallel t[s?\{ok.c!ok,ko\}] \]
What do we get by typing?

**Definition (p-Lock)**

$M'$ is a $p$-lock if the participant $p$ is willing to progress in $M'$ but cannot do that in any continuation of $M'$. 
What do we get by typing?

**Definition (Lock-freedom)**

$M$ is lock-free if, for each participant $p$,

$M \rightarrow^* M'$ implies $M'$ is not a $p$-lock.
What do we get by typing?

**Theorem (B. Dezani et al. FACS’22)**

If $\mathcal{M}$ is typable with a well-formed (bounded) global type, then $\mathcal{M}$ is lock-free.
Universal Declaration of system-Components’ Rights

- **Article 1** All system components must be developed equal in dignity and rights. They are endowed with communication capabilities and should interact towards one another in a spirit of cooperation.

- **Article 2** Any system component is entitled to all the good communication properties like lock freedom, without distinction of any kind, such as programming paradigm, language or interaction model.

- **Article 3** [...]


Universal Declaration of system-Components’ Rights

▶ **Article 1** All system components must be developed equal in dignity and rights. They are endowed with communication capabilities and should interact towards one another in a spirit of cooperation.

▶ **Article 2** Any system component is entitled to all the good communication properties like lock freedom, without distinction of any kind, such as programming paradigm, language or interaction model.

▶ **Article 3** [...]

Universal Declaration of system-Components’ Rights

- **Article 1** All system components must be developed equal in dignity and rights. They are endowed with communication capabilities and should interact towards one another in a spirit of cooperation.

- **Article 2** Any system component is entitled to all the good communication properties like lock freedom, without distinction of any kind, such as programming paradigm, language or interaction model.

- **Article 3** [...]


Universal Declaration of system-Components’ Rights

- **Article 1** All system components must be developed equal in dignity and rights. They are endowed with communication capabilities and should interact towards one another in a spirit of cooperation.

- **Article 2** Any system component is entitled to all the good communication properties like lock freedom, without distinction of any kind, such as programming paradigm, language or interaction model.

- **Article 3** [...]
Universal Declaration of system-Components’ Rights

▸ Article 1 All system components must be developed equal in dignity and rights. They are endowed with communication capabilities and should interact towards one another in a spirit of cooperation.

▸ Article 2 Any system component is entitled to all the good communication properties like lock freedom, without distinction of any kind, such as programming paradigm, language or interaction model.

▸ Article 3 […]
Article 1 All system components must be developed equal in dignity and rights. They are endowed with communication capabilities and should interact towards one another in a spirit of cooperation.

Article 2 Any system component is entitled to all the good communication properties like lock freedom without no distinction of any kind, such as programming paradigm, language or interaction model.

Article 3 [...]
Universal Declaration of system-Components’ Rights

- **Article 1** All system components must be developed in equal dignity and rights. They are endowed with communication capabilities and should interact towards one another in a spirit of cooperation.

- **Article 2** Any system component is entitled to all the good communication properties like lock freedom without distinction of any kind, such as programming paradigm, language or interaction model.

- **Article 3** [...]

![Image of Karl Marx](image1)

![Image of George Washington](image2)
Non egalitarian systems
Non egalitarian systems

Any client-server setting is biased:
Non egalitarian systems

Any client-server setting is biased:

The customer is always right.

— Harry Gordon Selfridge —
A non egalitarian system

A buyer can keep on adding goods - sold by a seller - in his shopping cart an unbounded number of times, until he decides to buy the shopping cart’s content. In the latter case, the seller informs the carrier for the shipment.
A non egalitarian system

A buyer can keep on adding goods - sold by a seller - in his shopping cart an unbounded number of times, until he decides to buy the shopping cart’s content. In the latter case, the seller informs the carrier for the shipment.

\[ M = b[B] \parallel s[S] \parallel c[C] \]

where
\[ B = s!\{\text{ADD}.B, \text{BUY}\} \]
\[ S = b?\{\text{ADD}.S, \text{BUY}.c!\text{SHIP}\} \]
\[ C = s?\text{SHIP}.C \]
A non egalitarian system

A buyer can keep on adding goods - sold by a seller - in his shopping cart an unbounded number of times, until he decides to buy the shopping cart’s content. In the latter case, the seller informs the carrier for the shipment.

\[ M = b[B] \parallel s[S] \parallel c[C] \]

where

\[ B = s!\{\text{ADD.}B, \text{BUY}\} \]
\[ S = b?\{\text{ADD.}S, \text{BUY.}c!\text{SHIP}\} \]
\[ C = s?\text{SHIP.}C \]

Not typable. In fact it is not c-lock free

\[ M \rightarrow^* b[0] \parallel s[0] \parallel c[C] \]
A non egalitarian system

A buyer can keep on adding goods - sold by a seller - in his shopping cart an unbounded number of times, until he decides to buy the shopping cart’s content. In the latter case, the seller informs the carrier for the shipment.

$$M = b[B] \parallel s[S] \parallel c[C]$$

where

$$B = s!\{\text{ADD}.B, \text{BUY}\}$$

$$S = b?\{\text{ADD}.S, \text{BUY}.c!\text{SHIP}\}$$

$$C = s?\text{SHIP}.C$$

Not typable. In fact it is not c-lock free

$$M \rightarrow^* b[0] \parallel s[0] \parallel c[C]$$

Do we actually care about s and c?
Typing non egalitarian systems

We are interested in b’s lock-freedom, not s’s and c’s

ICE’23: A type system such that if

\[ G \vdash \{ s, c \} \mathcal{M} \]

then lock-freedom ensured only for participants other than s and c. For our example this is possible for

\[ G = b \to s:\{ \text{ADD. } G, \text{ BUY} \} \]
Typing non egalitarian systems

We are interested in b’s lock-freedom, not s’s and c’s

ICE’23: A type system such that if

\[ G \vdash \{s, c\} M \]

then lock-freedom ensured only for participants other than s and c.

For our example this is possible for

\[ G = b \to s: \{\text{ADD. } G, \text{ BUY}\} \]
Typing non egalitarian systems

We are interested in b’s lock-freedom, not s’s and c’s

ICE’23: A type system such that if

\[ G \vdash \{s, c\} M \]

then lock-freedom ensured only for participants other than s and c. For our example this is possible for

\[ G = b \rightarrow s:\{\text{ADD. } G, \text{ BUY}\} \]
Typing non egalitarian systems

\[
\begin{array}{c}
\text{End} \vdash \emptyset \; p[0] \\
\text{G}_i \vdash \mathcal{P}_i \; p[P_i] \parallel q[Q_i] \parallel M \\
\left( \text{prt}(G_i) \cup \mathcal{P}_i \right) \setminus \{p, q\} = \text{prt}(M) \quad \forall i \in I \\
G \vdash p[q!\{\lambda_i. P_i\}_{i \in I}] \parallel q[p?\{\lambda_j. Q_j\}_{j \in J}] \parallel M \\
\end{array}
\]

\[G = p \rightarrow q : \{\lambda_i. G_i\}_{i \in I}\]
\[G \text{ is bounded}\]
\[\mathcal{P} = \bigcup_{i \in I} \mathcal{P}_i\]
\[I \subseteq J\]

[WEAK] \[
\begin{array}{c}
G \vdash \mathcal{P}_1 \; M_1 \\
G \vdash \mathcal{P}_1 \cup \mathcal{P}_2 \; M_1 \parallel M_2 \\
\end{array}
\]
\[\mathcal{P}_2 = \text{prt}(M_2) \neq \emptyset\]
Typing non egalitarian systems

\[
\text{[End]} \quad \frac{\text{End} \vdash \emptyset \ p[0]}{\text{[End]}}
\]

\[
G_i \vdash P_i \ | \ q[Q_i] \ | \ M
\]

\[
(\text{prt}(G_i) \cup P_i) \setminus \{p, q\} = \text{prt}(M) \quad \forall i \in I
\]

\[
G \vdash p[q!\{\lambda_i. P_i\}_{i \in I}] \ | \ q[p?\{\lambda_j. Q_j\}_{j \in J}] \ | \ M
\]

G = p \rightarrow q : \{\lambda_i. G_i\}_{i \in I}

G is bounded

\[\mathcal{P} = \bigcup_{i \in I} P_i\]

I \subseteq J

\[\text{[WEAK]}\]

\[
G \vdash \mathcal{P}_1 \ M_1, M_2
\]

\[
G \vdash \mathcal{P}_1 \cup \mathcal{P}_2 \ M_1 \ | \ M_2\]

\[\mathcal{P}_2 = \text{prt}(M_2) \neq \emptyset\]
Typing non egalitarian systems

\[ \text{End} \quad \frac{\text{End} \vdash \emptyset \; p[0]}{\text{End}} \]

\[
G_i \vdash \mathcal{P}_i \; p[P_i] \parallel q[Q_i] \parallel M
\]

\[
(prt(G_i) \cup \mathcal{P}_i) \setminus \{p, q\} = prt(M) \quad \forall i \in I
\]

\[
G \vdash \mathcal{P} \; p[q!\{\lambda i. P_i\}_{i \in I}] \parallel q[p?\{\lambda j. Q_j\}_{j \in J}] \parallel M
\]

\[
G = p \rightarrow q : \{\lambda i. G_i\}_{i \in I}
\]

\[
\text{G is bounded}
\]

\[
\mathcal{P} = \bigcup_{i \in I} \mathcal{P}_i
\]

\[
I \subseteq J
\]

\[ [\text{WEAK}] \quad \frac{G \vdash \mathcal{P}_1 \; M_1}{G \vdash \mathcal{P}_1 \cup \mathcal{P}_2 \; M_1 \parallel M_2} \quad \mathcal{P}_2 = \text{prt}(M_2) \neq \emptyset \]
Typing non egalitarian systems

\[ \text{End} \vdash \emptyset \ p[0] \]

\[
\begin{align*}
G_i & \vdash \mathcal{P}_i \ p[P_i] \parallel q[Q_i] \parallel \mathcal{M} \\
(prt(G_i) \cup \mathcal{P}_i) \setminus \{p, q\} & = \text{prt}(\mathcal{M}) \ \forall i \in I \\
G & \vdash \mathcal{P} \ p[q!\{\lambda_i. P_i\}_{i \in I}] \parallel q[p?\{\lambda_j. Q_j\}_{j \in J}] \parallel \mathcal{M}
\end{align*}
\]

\[
G = p \rightarrow q : \{\lambda_i. G_i\}_{i \in I} \\
\text{G is bounded} \\
\mathcal{P} = \bigcup_{i \in I} \mathcal{P}_i \\
I \subseteq J
\]

\[
\begin{align*}
\text{WEAK} & \quad \frac{G \vdash \mathcal{P}_1 \ \mathcal{M}_1}{G \vdash \mathcal{P}_1 \cup \mathcal{P}_2 \ \mathcal{M}_1 \parallel \mathcal{M}_2} \\
\mathcal{P}_2 & = \text{prt}(\mathcal{M}_2) \neq \emptyset
\end{align*}
\]
Typing non egalitarian systems

**Theorem (Classist lock-freedom)**

*If $G \vdash_{\mathcal{P}} \mathcal{M}$ then $\mathcal{M}$ is $p$-lock free only if $p \notin \mathcal{P}$.*
Typing non egalitarian systems

\[ D = \frac{\text{End} \vdash \emptyset b[0]}{\text{End} \vdash \{s,c\} b[0] || s[c!\text{SHIP}] || c[C]} \frac{[\text{WEAK}]}{G \vdash \{s,c\} b[B] || s[b?\{\text{ADD}.S, \text{PAY}.c!\text{SHIP}\}] || c[s?\text{SHIP}.C]} \]

where \( G = b \rightarrow s:\{\text{ADD}.G, \text{BUY}\} \)

\( B = s!\{\text{ADD}.B, \text{PAY}\} \)

\( S = b?\{\text{ADD}.S, \text{BUY}.c!\text{SHIP}\} \)

\( C = s?\text{SHIP}.C \)

Hence the Buyer-Seller-Carrier system is \( b \)-lock free
Typing non egalitarian systems

\[ \mathcal{D} = \begin{array}{c}
\text{End} \vdash \emptyset b[0] \\
\text{End} \vdash \{s,c\} b[0] \parallel s[c!\text{ship}] \parallel c[C] \\
G \vdash \{s,c\} b[B] \parallel s[b\{\text{ADD}.S, \text{PAY}.c!\text{ship}\}] \parallel c[s?\text{ship}.C]
\end{array} \quad [\text{WEAK}] \\

\text{where } G = b \rightarrow s:\{\text{ADD}.G, \text{BUY}\} \\
B = s!\{\text{ADD}.B, \text{PAY}\} \\
S = b?\{\text{ADD}.S, \text{BUY}.c!\text{ship}\} \\
C = s?\text{ship}.C \\

Hence the Buyer-Seller-Carrier system is \textit{b-lock free}.
Typing non egalitarian systems

\[ \mathcal{D} = \begin{array}{c}
\text{End} \vdash \emptyset \ b[0] \\
\text{End} \vdash \{s, c\} \ b[0] \parallel s[c!\text{SHIP}] \parallel c[C] \\
G \vdash \{s, c\} \ b[B] \parallel s[b\{\text{ADD}.S, \text{PAY}.c!\text{SHIP}\}] \parallel c[s?\text{SHIP}.C]
\end{array} \]  

where \( G = b \rightarrow s:\{\text{ADD}. G, \text{BUY}\} \)

\( B = s!\{\text{ADD}.B, \text{PAY}\} \)

\( S = b?\{\text{ADD}.S, \text{BUY}.c!\text{SHIP}\} \)

\( C = s?\text{SHIP}.C \)

Hence the Buyer-Seller-Carrier system is \( b \)-lock free
Ongoing work
Ongoing work

- Overshooting:
Ongoing work

- Overshooting: Well-formedness condition for global types does actually ensure more than needed (strong $p$-lock freedom)
Ongoing work

- Overshooting: Well-formedness condition for global types does actually ensure more than needed (strong p-lock freedom)
- Typing non egalitarian asynchronous systems.
Ongoing work

- Overshooting: Well-formedness condition for global types does actually ensure more than needed (strong p-lock freedom)
- Typing non egalitarian asynchronous systems.