Session-Typed Concurrent Programming

Lecture 3

Stephanie Balzer
Carnegie Mellon University

OPLSS 2021
June 25, 2021
Today’s lecture
Today’s lecture

Recap

- Type system and dynamics for the intuitionistic linear session types language SILL
- Curry-Howard correspondence
- SILL readily guarantees session fidelity and deadlock-freedom
Today’s lecture

Recap

• Type system and dynamics for the intuitionistic linear session types language SILL
• Curry-Howard correspondence
• SILL readily guarantees session fidelity and deadlock-freedom

Next

• Extend SILL with persistent truth (of course!)
• Then, switch gears and introduce shared session types
Follow-up on Slack
Follow-up on Slack

\[
\Delta_1 \vdash P :: (x : A) \quad \Delta_2, x : A \vdash Q :: (z : C)
\]

\[
\Delta_1, \Delta_2 \vdash x \leftarrow P; Q :: (z : C)
\]
Follow-up on Slack

\[
\frac{\Delta_1 \vdash P :: (x : A) \quad \Delta_2, x : A \vdash Q :: (z : C)}{\Delta_1, \Delta_2 \vdash x \leftarrow P; Q :: (z : C)} \text{ Cut}
\]

\[\text{(D-Cut)} \quad \text{proc}(c, x \leftarrow P_x; Q_x) \quad \longrightarrow \quad \text{proc}(a, [a/x] P_x), \text{proc}(c, [a/x] Q_x) \quad \text{(a fresh)}\]
Follow-up on Slack

\[
\begin{align*}
\Delta_1 \vdash P :: (x : A) & \quad \Delta_2, x : A \vdash Q :: (z : C) \\
\Delta_1, \Delta_2 \vdash x \leftarrow P; Q :: (z : C) & \quad \text{Cut}
\end{align*}
\]

\[(\text{D-Cut}) \quad \begin{array}{l}
\text{proc}(c, x \leftarrow P_x; Q_x) \\
\rightarrow \text{proc}(a, [a/x] P_x), \text{proc}(c, [a/x] Q_x) \quad \text{(a fresh)}
\end{array}\]

\[
\begin{array}{c}
c
\end{array}
\]

\[
S = x \leftarrow P_x; Q_x
\]
Follow-up on Slack

$$\Delta_1 \vdash P :: (x : A) \quad \Delta_2, x : A \vdash Q :: (z : C)$$
$$\Delta_1, \Delta_2 \vdash x \leftarrow P; Q :: (z : C) \quad \text{Cut}$$

(D-Cut) \quad \text{proc} (c, x \leftarrow P_x; Q_x) \\
\quad \quad \rightarrow \quad \text{proc} (a, [a/x] P_x), \text{proc} (c, [a/x] Q_x) \quad \text{(a fresh)}

$$S = x \leftarrow P_x; Q_x$$
Follow-up on Slack

\[
\begin{align*}
\Delta_1 \vdash P :: (x : A) \quad &\quad \Delta_2, x : A \vdash Q :: (z : C) \\
\Delta_1, \Delta_2 \vdash x \leftarrow P; Q :: (z : C) &\qquad \text{Cut}
\end{align*}
\]

\[
\begin{align*}
(D\text{-}Cut) \quad &\quad \text{proc}(c, x \leftarrow P_x; Q_x) \\
&\quad \rightarrow \quad \text{proc}(a, [a/x] P_x), \text{proc}(c, [a/x] Q_x) \quad \text{(a fresh)}
\end{align*}
\]

\[
\begin{aligned}
S &= x \leftarrow P_x; Q_x \\
S' &= [a/x] Q_x \\
T &= [a/x] P_x
\end{aligned}
\]
Follow-up on Slack

\[
\Delta_1 \vdash P :: (x : A) \quad \Delta_2, x : A \vdash Q :: (z : C) \\
\Delta_1, \Delta_2 \vdash x \leftarrow P; Q :: (z : C) \tag{Cut}
\]

\[(D-Cut) \quad \text{proc}(c, x \leftarrow P_x; Q_x) \quad \longrightarrow \quad \text{proc}(a, [a/x] P_x), \text{proc}(c, [a/x] Q_x) \quad \text{(a fresh)}
\]

\[
\begin{array}{c}
\text{c} \\
S
\end{array} \quad \xrightarrow{\text{c}} \quad \begin{array}{c}
\text{c} \\
S'
\end{array}
\]

\[
S = x \leftarrow P_x; Q_x \\
S' = [a/x] Q_x \\
T = [a/x] P_x
\]
Intuitionistic linear logic session types with !
Of course!
Of course!

one connective from linear logic still missing: persistent truth
Of course!

one connective from linear logic still missing: persistent truth

Types:

\[ A, B \triangleq A \otimes B \]  multiplicative conjunction  “channel output”
\[ A \multimap B \]  multiplicative implication  “channel input”
\[ A \& B \]  additive conjunction  “external choice”
\[ A \oplus B \]  additive disjunction  “internal choice”
\[ 1 \]  unit for \( \otimes \)  “termination”
Of course!

one connective from linear logic still missing: persistent truth

Types:

\[ A, B \triangleq A \otimes B \quad \text{multiplicative conjunction} \quad \text{“channel output”} \]
\[ A \multimap B \quad \text{multiplicative implication} \quad \text{“channel input”} \]
\[ A \& B \quad \text{additive conjunction} \quad \text{“external choice”} \]
\[ A \oplus B \quad \text{additive disjunction} \quad \text{“internal choice”} \]
\[ 1 \quad \text{unit for } \otimes \quad \text{“termination”} \]
\[ !A \quad \text{”of course”, persistent truth} \quad \text{“replication”} \]
Of course!

One connective from linear logic still missing: persistent truth

Types:

\[ A, B \triangleq A \otimes B \] multiplicative conjunction “channel output”
\[ A \multimap B \] multiplicative implication “channel input”
\[ A \& B \] additive conjunction “external choice”
\[ A \oplus B \] additive disjunction “internal choice”
\[ 1 \] unit for \( \otimes \) “termination”
\[ !A \] ”of course”, persistent truth “replication”

A process of type \(!A\) can be used arbitrarily often, i.e., can have any number of clients
Of course!

What is the computational meaning of “of course”? 
Of course!

What is the computational meaning of “of course”?
Of course!

What is the computational meaning of “of course”?

linear process
Of course!

What is the computational meaning of “of course”?

linear process

persistent process
Of course!

What is the computational meaning of “of course”? 

\[ u: A \]

\[ P \]
Of course!

What is the computational meaning of “of course”?

obtain a linear copy $P'$ of unrestricted process $P$
Of course!

What is the computational meaning of “of course”?

obtain a linear copy $P'$ of unrestricted process $P$
Of course!

What is the computational meaning of “of course”?

Obtain a linear copy $P'$ of unrestricted process $P$. 

Diagram: 
- $Q \xrightarrow{u:A} P$ 
- $Q \xrightarrow{u:A} P'$ 
  - $P'$ is the linear copy of $P$. 
  - The diagram shows the process $P$ being copied and then a linear copy is obtained.
Of course!

What is the computational meaning of “of course”?

obtain a linear copy \( P' \) of unrestricted process \( P \)
Of course!

What is the computational meaning of “of course”? 

obtain a linear copy $P'$ of unrestricted process $P$

corresponds to replication in the pi-calculus
Of course!

What is the computational meaning of “of course”?

- obtain a linear copy $P'$ of unrestricted process $P$
- corresponds to replication in the pi-calculus
- let’s look at typing rules and dynamics
Of course!

What is the computational meaning of “of course”?

![Diagram](https://via.placeholder.com/150)

- obtain a linear copy $P'$ of unrestricted process $P$
- corresponds to replication in the pi-calculus
- let’s look at typing rules and dynamics

(*) copy rule operates on structural context, so it should be $u: A$ because $A$ is judgmentally persistent
Of course!

Types:

\[ A, B \triangleq A \otimes B \quad \text{multiplicative conjunction} \quad \text{“channel output”} \]
\[ A \rightarrow \circ B \quad \text{multiplicative implication} \quad \text{“channel input”} \]
\[ A \& B \quad \text{additive conjunction} \quad \text{“external choice”} \]
\[ A \oplus B \quad \text{additive disjunction} \quad \text{“internal choice”} \]
\[ 1 \quad \text{unit for } \otimes \quad \text{“termination”} \]
\[ !A \quad 
\text{"of course", persistent truth} \quad \text{“replication”} \]
Of course!

Types:

\[ A, B \triangleq A \otimes B \] multiplicative conjunction “channel output”
\[ A \to B \] multiplicative implication “channel input”
\[ A \& B \] additive conjunction “external choice”
\[ A \oplus B \] additive disjunction “internal choice”
\[ 1 \] unit for \( \otimes \) “termination”
\[ !A \] ”of course”, persistent truth “replication”

Typing judgment:

\[ \Psi; \Delta \vdash P :: (x : A) \]
Of course!

Types:

\[ A, B \triangleq A \otimes B \quad \text{multiplicative conjunction} \quad \text{“channel output”} \]
\[ A \rightarrow B \quad \text{multiplicative implication} \quad \text{“channel input”} \]
\[ A \& B \quad \text{additive conjunction} \quad \text{“external choice”} \]
\[ A \oplus B \quad \text{additive disjunction} \quad \text{“internal choice”} \]
\[ 1 \quad \text{unit for } \otimes \quad \text{“termination”} \]
\[ !A \quad \text{”of course”, persistent truth} \quad \text{“replication”} \]

Typing judgment:

\[ \Psi; \Delta \vdash P :: (x : A) \]
Of course!

Types:

\[
\begin{align*}
A, B & \triangleq A \otimes B & \text{multiplicative conjunction} & \text{“channel output”} \\
A \multimap B & \text{multiplicative implication} & \text{“channel input”} \\
A \& B & \text{additive conjunction} & \text{“external choice”} \\
A \oplus B & \text{additive disjunction} & \text{“internal choice”} \\
1 & \text{unit for } \otimes & \text{“termination”} \\
!A & \text{”of course”, persistent truth} & \text{“replication”}
\end{align*}
\]

Typing judgment:

\[\Psi; \Delta \vdash P :: (x : A)\]
Of course!

Types:

\[ A, B \triangleq A \otimes B \] multiplicative conjunction  
“channel output”

\[ A \multimap B \] multiplicative implication  
“channel input”

\[ A \& B \] additive conjunction  
“external choice”

\[ A \oplus B \] additive disjunction  
“internal choice”

\[ 1 \] unit for \( \otimes \)  
“termination”

\[ !A \] ”of course”, persistent truth  
“replication”

Typing judgment:

\[ \Psi; \Delta \vdash P :: (x : A) \]

persistent channels

structural context, i.e., permits weakening and contraction
Of course!

Types:

\[
    A, B \triangleq A \otimes B \quad \text{multiplicative conjunction} \quad \text{“channel output”}
\]

\[
    A \rightarrow B \quad \text{multiplicative implication} \quad \text{“channel input”}
\]

\[
    A \& B \quad \text{additive conjunction} \quad \text{“external choice”}
\]

\[
    A \oplus B \quad \text{additive disjunction} \quad \text{“internal choice”}
\]

\[
    1 \quad \text{unit for } \otimes \quad \text{“termination”}
\]

\[
    !A \quad \text{”of course”, persistent truth} \quad \text{“replication”}
\]

Typing judgment:

\[
    \Psi; \Delta \vdash P :: (x : A)
\]
Types:

\[ A, B \triangleq A \otimes B \] multipliclicative conjunction \hspace{2cm} “channel output”
\[ A \multimap B \] multiplicative implication \hspace{2cm} “channel input”
\[ A \& B \] additive conjunction \hspace{2cm} “external choice”
\[ A \oplus B \] additive disjunction \hspace{2cm} “internal choice”
\[ 1 \] unit for \( \otimes \) \hspace{2cm} “termination”
\[ !A \] ”of course”, persistent truth \hspace{2cm} “replication”

Typing judgment:

\[ \Psi; \Delta \vdash P :: (x : A) \]
Of course!

Types:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A, B$</td>
<td>multiplicative conjunction</td>
<td>$A \otimes B$</td>
<td>“channel output”</td>
</tr>
<tr>
<td>$A \multimap B$</td>
<td>multiplicative implication</td>
<td>$A \multimap B$</td>
<td>“channel input”</td>
</tr>
<tr>
<td>$A &amp; B$</td>
<td>additive conjunction</td>
<td>$A &amp; B$</td>
<td>“external choice”</td>
</tr>
<tr>
<td>$A \oplus B$</td>
<td>additive disjunction</td>
<td>$A \oplus B$</td>
<td>“internal choice”</td>
</tr>
<tr>
<td>$1$</td>
<td>unit for $\otimes$</td>
<td></td>
<td>“termination”</td>
</tr>
<tr>
<td>$!A$</td>
<td>“of course”, persistent truth</td>
<td></td>
<td>“replication”</td>
</tr>
</tbody>
</table>

Typing judgment:

$$\Psi; \Delta \vdash P :: (x : A)$$
Of course!

Types:

\[ A, B \triangleq A \otimes B \quad \text{multiplicative conjunction} \quad \text{“channel output”} \]
\[ A \rightarrow B \quad \text{multiplicative implication} \quad \text{“channel input”} \]
\[ A \& B \quad \text{additive conjunction} \quad \text{“external choice”} \]
\[ A \oplus B \quad \text{additive disjunction} \quad \text{“internal choice”} \]
\[ 1 \quad \text{unit for } \otimes \quad \text{“termination”} \]
\[ !A \quad \text{“of course”, persistent truth} \quad \text{“replication”} \]

Typing judgment:

\[ \Psi; \Delta \vdash P :: (x : A) \]
\[ \Psi = u_1 : B_1, \ldots, u_n : B_n \]
Of course!

Types:

\[
A, B \quad \triangleq \quad A \otimes B \quad \text{multiplicative conjunction} \\
A \rightarrow B \quad \text{multiplicative implication} \\
A \& B \quad \text{additive conjunction} \\
A \oplus B \quad \text{additive disjunction} \\
1 \quad \text{unit for } \otimes \\
!A \quad \text{"of course", persistent truth}
\]

“channel output”
“channel input”
“external choice”
“internal choice”
“termination”
“replication”

Typing judgment:

\[
\Psi; \Delta \vdash P :: (x : A)
\]

\[
\Psi = u_1 : B_1, \ldots, u_n : B_n
\]

dyadic formulation

implicitly !-typed
Of course!

Types:

\[ A, B \triangleq A \otimes B \]  multiplicative conjunction  “channel output”

\[ A \rightarrow B \]  multiplicative implication  “channel input”

\[ A \& B \]  additive conjunction  “external choice”

\[ A \oplus B \]  additive disjunction  “internal choice”

\( \text{1} \)  unit for \( \otimes \)

\( !A \)  ”of course”, persistent truth  “termination”

Typing judgment:

\[ \Psi; \Delta \vdash P :: (x : A) \]

\[ \Psi = u_1 : B_1, \ldots, u_n : B_n \]

Persistent channels in \( \Delta \) are of type \( !A \)

Dyadic formulation

Implicitly \( ! \)-typed
Judgmental rule copy
Judgmental rule copy

Typing rule:

\[
\frac{\Psi, u : A; \Delta, x : A \vdash Q_x :: (z : C')}{
\Psi, u : A; \Delta \vdash \text{send } u (\text{new } x); Q_x :: (z : C') \quad \text{copy}
}\]
Judgmental rule copy

Typing rule:

\[
\begin{align*}
\Psi, \: u : A; \: \Delta, \: x : A & \vdash Q_x :: (z : C') \\
\Psi, \: u : A; \: \Delta & \vdash \text{send } u (\text{new } x); \: Q_x :: (z : C')
\end{align*}
\]

obtain a linear copy of a persistent server
Judgmental rule copy

Typing rule: contraction!

\[
\begin{align*}
\Psi, u : A; \Delta, x : A & \vdash Q_x :: (z : C') \\
\Psi, u : A; \Delta & \vdash \text{send } u (\text{new } x); Q_x :: (z : C') & \text{copy}
\end{align*}
\]

obtain a linear copy of a persistent server
Judgmental rule copy

Typing rule:

$$
\frac{\Psi, u : A; \Delta, x : A \vdash Q_x :: (z : C')}{\Psi, u : A; \Delta \vdash \text{send} \ u \ (\text{new} \ x) ; Q_x :: (z : C')} \quad \text{copy}
$$

obtain a linear copy of a persistent server
Judgmental rule copy

Typing rule:

\[
\frac{
\Psi, u : A; \Delta, x : A \vdash Q_x :: (z : C)
}{
\Psi, u : A; \Delta \vdash \text{send } u (\text{new } x); Q_x :: (z : C)
} \quad \text{copy}
\]

Dynamics:

\[
(D\text{-copy}) \quad \text{!proc}(u, x \leftarrow \text{recv } u; P_x), \text{proc}(c, \text{send } u (\text{new } x); Q_x) \\
\rightarrow \text{proc}(a, [a/x] P_x), \text{proc}(c, [a/x] Q_x) \quad \text{(a fresh)}
\]
Judgmental rule copy

Typing rule:

\[
\begin{align*}
\Psi, u : A; \Delta, x : A & \vdash Q_x :: (z : C) \\
\Psi, u : A; \Delta & \vdash \text{send } u \ (\text{new } x); Q_x :: (z : C)
\end{align*}
\]

obtain a linear copy of a persistent server

Dynamics:

\[
(D\text{-}copy) \quad !\text{proc}(u, x \leftarrow \text{recv } u; P_x), \ \text{proc}(c, \text{send } u \ (\text{new } x); Q_x) \\
\longrightarrow \text{proc}(a, [a/x] P_x), \ \text{proc}(c, [a/x] Q_x) \quad \text{(a fresh)}
\]

persistent!
Judgmental rule copy

Typing rule:

\[
\Psi, u : A; \Delta, x : A \vdash Q_x :: (z : C') \\
\Psi, u : A; \Delta \vdash \text{send } u (\text{new } x); Q_x :: (z : C')
\]

obtain a linear copy of a persistent server

Dynamics:

\[(D\text{-copy}) \quad \begin{array}{l}
!\text{proc}(u, x \leftarrow \text{recv } u; P_x), \text{proc}(c, \text{send } u (\text{new } x); Q_x) \\
\rightarrow \text{proc}(a, [a/x] P_x), \text{proc}(c, [a/x] Q_x) \quad (\text{a fresh})
\end{array}\]

persistent!

remains available in post-state
Judgmental rule cut!
Judgmental rule cut!

Typing rule:

\[
\frac{
\Psi; \cdot \vdash P_x :: (x : A) \quad \Psi, u : A; \Delta \vdash Q_u :: (z : C)
}{
\Psi; \Delta \vdash u \leftarrow ! (x \leftarrow \text{recv } u ; P_x) ; Q_u :: (z : C)
}\text{ cut!}
\]
Judgmental rule cut!

Typing rule:

\[
\begin{align*}
\Psi; \cdot & \vdash P_x :: (x : A) & \Psi, u : A; \Delta & \vdash Q_u :: (z : C) \\
\Psi; \Delta & \vdash u \leftarrow !(x \leftarrow \text{recv } u; P_x); Q_u :: (z : C) & \text{cut!}
\end{align*}
\]

spawning a persistent server
Judgmental rule cut!

Typing rule:

\[
\frac{\Psi; \cdot \vdash P_x :: (x : A) \quad \Psi, u : A; \Delta \vdash Q_u :: (z : C)}{\Psi; \Delta \vdash u \leftarrow !(x \leftarrow \text{recv } u; P_x); Q_u :: (z : C)} \quad \text{cut!}
\]

spawning a persistent server

Dynamics:

(D-cut!) \quad \text{proc}(c, u \leftarrow !(x \leftarrow \text{recv } u; P_x); Q_u)

\rightarrow !\text{proc}(a, x \leftarrow \text{recv } a; P_x), \text{proc}(c, [a/u] Q_u) \quad \text{(a fresh)}
Rules for of course
Rules for of course

Typing rule:

\[
\begin{align*}
\Psi; \cdot & \vdash Py :: (y : A) \\
\Psi; \cdot & \vdash \text{send } x (\text{new } u); ! (y \leftarrow \text{recv } u; Py) :: (x : ! A) \\
\Psi, u : A; \Delta & \vdash Qu :: (z : C) \\
\Psi; \Delta, x : ! A & \vdash u \leftarrow \text{recv } x; Qu :: (z : C)
\end{align*}
\]
Rules for of course

Typing rule:

\[
\frac{
\Psi; \cdot \vdash P_y :: (y : A)
}{
\Psi; \cdot \vdash \text{send } x \text{ (new } u \text{); !(y } \leftarrow \text{ recv } u; P_y \text{)} :: (x :!A) \quad !_R
}\]

\[
\frac{
\Psi, u : A; \Delta \vdash Q_u :: (z : C)
}{
\Psi; \Delta, x :!A \vdash u \leftarrow \text{recv } x; Q_u :: (z : C) \quad !_L
}\]

spawning a persistent server
Rules for of course

Typing rule:

\[
\begin{align*}
\Psi; \cdot \vdash P_y &:: (y : A) \\
\Psi; \cdot \vdash \text{send } x (\text{new } u); !(y \leftarrow \text{recv } u; P_y) &:: (x :!A) \quad !_R \\
\Psi, u : A; \Delta \vdash Q_u &:: (z : C) \\
\Psi; \Delta, x :!A \vdash u \leftarrow \text{recv } x; Q_u &:: (z : C) \quad !_L
\end{align*}
\]

spawning a persistent server

Dynamics:

\[(D-!) \quad \text{proc}(a, \text{send } a (\text{new } u); !(y \leftarrow \text{recv } u; P_y)), \quad \text{proc}(c, u \leftarrow \text{recv } a; Q_u) \]
\[\rightarrow !\text{proc}(b, y \leftarrow \text{recv } b; P_y), \quad \text{proc}(c, [b/u] Q_u) \quad \text{(b fresh)}\]
Taking stock
Taking stock

Replication — clients are shielded from each others effects
Taking stock

Replication — clients are shielded from each others effects
Taking stock

Replication — clients are shielded from each others effects
Taking stock

Replication — clients are shielded from each other's effects

Diagram:

- P'
- Q_1
- P''
- Q_2
- P

Arrows indicate the direction of interaction.
Taking stock

Replication — clients are shielded from each others effects

any communication of one client with its copy of P will not affect the private copies of P of other clients
Taking stock

Replication — clients are shielded from each others effects

any communication of one client with its copy of P will not affect the private copies of P of other clients

for some applications this copying semantics is appropriate
Taking stock

Replication — clients are shielded from each others effects

any communication of one client with its copy of P will not affect the private copies of P of other clients

for some applications this copying semantics is appropriate

other applications need a true sharing semantics
Taking stock

Replication — clients are shielded from each others effects

any communication of one client with its copy of P will not affect the private copies of P of other clients

for some applications this copying semantics is appropriate

other applications need a true sharing semantics

let’s explore next!
Manifest sharing
Manifest sharing — key ideas
Manifest sharing — key ideas

permit aliases, rather than ruling them out
Manifest sharing — key ideas

- permit aliases, rather than ruling them out
- to guarantee preservation
Manifest sharing — key ideas

- permit aliases, rather than ruling them out
- to guarantee preservation
- exclusive access required prior any communication
Manifest sharing — key ideas

- permit aliases, rather than ruling them out
- to guarantee preservation
- exclusive access required prior any communication
- relinquish exclusive access in consistent state
Manifest sharing — key ideas

- permit aliases, rather than ruling them out to guarantee preservation
- exclusive access required prior any communication
- relinquish exclusive access in consistent state
- manifest these ideas in type structure
Manifest sharing — key ideas

- permit aliases, rather than ruling them out to guarantee preservation
- exclusive access required prior any communication
- relinquish exclusive access in consistent state
- manifest these ideas in type structure

Copying versus sharing semantics
Copying versus sharing semantics

Copying semantics

Q₁

Q₂

u!:A

P
Copying versus sharing semantics

Copying semantics

Sharing semantics
Copying versus sharing semantics

Copying semantics

Sharing semantics

persistent
Copying versus sharing semantics

Copying semantics

Sharing semantics
Copying versus sharing semantics

Copying semantics

Sharing semantics

Q₁

Q₂

P

u:A

persistent

Q₁

Q₂

P

x:Aₛ

shared
Copying versus sharing semantics

Copying semantics

Sharing semantics
Copying versus sharing semantics

Copying semantics

Sharing semantics
Copying versus sharing semantics

Copying semantics

Sharing semantics
Copying versus sharing semantics

Copying semantics

Sharing semantics
Copying versus sharing semantics

Copying semantics

Sharing semantics

private linear copy
Copying versus sharing semantics

Copying semantics

Sharing semantics

private linear copy
Copying versus sharing semantics

Copying semantics

Sharing semantics

private linear copy

private linear channel
Key idea 1: acquire-release
Key idea 1: acquire-release
Key idea 1: acquire-release
Key idea 1: acquire-release
Key idea 1: acquire-release
Key idea 1: acquire-release

Legend:  
- Linear channel
- Linear process
- Shared channel
- Shared process
Key idea 1: acquire-release

Legend:
- • linear channel
- ➡ linear process
- ••• shared channel
- ••••• shared process

Both clients contend for communicating with P.
Key idea 1: acquire-release

Legend:
- → linear channel
- ○ linear process
- ➔ shared channel
- ○ shared process

both clients contend for communicating with P

\[
x : A_s
\]

Q₁

Q₂

acq
Key idea 1: acquire-release
Key idea 1: acquire-release

Legend:
- → linear channel
- ← shared channel
- Ⓚ linear process
- Ⓚ shared process
Key idea 1: acquire-release

Q1 has exclusive access to P

Legend:
- ➔ linear channel
- blue circle linear process
- ➠ shared channel
- red circle shared process
Key idea 1: acquire-release

Q1 has exclusive access to P

Q1 communicates along private channel y

Legend:

- linear channel
- linear process
- shared channel
- shared process
Key idea 1: acquire-release
Key idea 1: acquire-release

Q1 relinquishes exclusive access to P
Key idea 1: acquire-release

Q1 relinquishes exclusive access to P
Key idea 1: acquire-release

Q1 relinquishes exclusive access to P
Key idea 1: acquire-release

Legend:
- linear channel
- linear process
- shared channel
- shared process
Key idea 1: acquire-release

Q1 must contend again for P
Key idea 1: acquire-release

Q1 must contend again for P

only acquire messages can be sent along shared channels

Legend:
- ➔ linear channel
- ⬤ linear process
- ➔ shared channel
-🔴 shared process
Key idea 2: manifest acquire-release in types
Key idea 2: manifest acquire-release in types

Observation:
Key idea 2: manifest acquire-release in types

Observation:

processes are at one of two modes: either linear or shared
Key idea 2: manifest acquire-release in types

Observation:

processes are at one of two modes: either linear or shared

Adjoint stratification of session types:
Key idea 2: manifest acquire-release in types

Observation:

- processes are at one of two modes: either linear or shared

Adjoint stratification of session types:

- stratify session types into a linear and shared layer, s.t. $S > L$
Key idea 2: manifest acquire-release in types

Observation:
processes are at one of two modes: either linear or shared

Adjoint stratification of session types:
stratify session types into a linear and shared layer, s.t. $S > L$

\[
\begin{align*}
A_S & \triangleq \\
A_L, B_L & \triangleq \bigoplus \{ \overline{l : A_L} \} \mid A_L \otimes B_L \mid 1 \mid \\
\& \{ \overline{l : A_L} \} \mid A_L \multimap B_L
\end{align*}
\]
Key idea 2: manifest acquire-release in types

Observation:

- processes are at one of two modes: either linear or shared

Adjoint stratification of session types:

- stratify session types into a linear and shared layer, s.t. $S > L$

Weakening
contraction

\[
A_S \triangleq \begin{cases} 
A_L, B_L & \Delta \equiv \oplus\{l : A_L\} \mid A_L \otimes B_L \mid 1 \mid \\
\&\{l : A_L\} \mid A_L \leftrightarrow B_L \end{cases}
\]
Key idea 2: manifest acquire-release in types

Observation:

- processes are at one of two modes: either linear or shared

Adjoint stratification of session types:

- stratify session types into a linear and shared layer, s.t. $S > L$
- connect layers with modalities going back and forth

\[
\begin{align*}
A_S & \triangleq \\
A_L, B_L & \triangleq \bigoplus \{l : A_L\} \mid A_L \otimes B_L \mid 1 \mid \\
& \quad \& \{l : A_L\} \mid A_L \multimap B_L
\end{align*}
\]
Key idea 2: manifest acquire-release in types

Observation:

processes are at one of two modes: either linear or shared

Adjoint stratification of session types:

stratify session types into a linear and shared layer, s.t. \( S > L \)

connect layers with modalities going back and forth

weakening
contraction

\[
\begin{align*}
A_S & \triangleq \uparrow^S L A_L \\
A_L, B_L & \triangleq \bigoplus \{l : A_L\} | A_L \otimes B_L | 1 | \\
& \quad \& \{l : A_L\} | A_L \rightarrow B_L | \downarrow^S L A_S
\end{align*}
\]
Key idea 2: manifest acquire-release in types

Observation:

- processes are at one of two modes: either linear or shared

Adjoint stratification of session types:

- stratify session types into a linear and shared layer, s.t. $S > L$
- connect layers with modalities going back and forth

Weakening contraction:

\[ A_S \triangleq \uparrow^S L A_L \]

\[ A_L, B_L \triangleq \oplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \]

\[ \& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S L A_S \]

Support of sending shared channels along linear channels
Key idea 2: manifest acquire-release in types

Observation:
- Processes are at one of two modes: either linear or shared.

Adjoint stratification of session types:
- Stratify session types into a linear and shared layer, such that $S > L$.
- Connect layers with modalities going back and forth.

Weakening and contraction:
- $A_S \triangleq \uparrow^S_L A_L$
- $A_L, B_L \triangleq \bigoplus \{\overline{l : A_L}\mid A_L \otimes B_L \mid 1 \mid \exists x: A_S. B_L \mid \bigland\{\overline{l : A_L}\mid A_L \rightarrow B_L \mid \downarrow^S_L A_S \mid \Pi x: A_S. B_L\}$

Support of sending shared channels along linear channels:
Example: shared queue

What should be the type of a shared queue?
Example: shared queue

What should be the type of a shared queue?

\[
A_S \triangleq \uparrow^S_L A_L \\
A_L, B_L \triangleq \oplus \{l : A_L\} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S . B_L \\
& \& \{l : A_L\} \mid A_L \rightarrow B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S . B_L
\]
Example: shared queue

What should be the type of a shared queue?

<table>
<thead>
<tr>
<th>weakening</th>
<th>contraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$A_S$</td>
<td>$\uparrow^S_L A_L$</td>
</tr>
<tr>
<td>$A_L, B_L$</td>
<td>$\oplus{l : A_L} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S . B_L \mid$</td>
</tr>
<tr>
<td></td>
<td>$&amp;{l : A_L} \mid A_L \rightarrow B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S . B_L$</td>
</tr>
</tbody>
</table>

queue $A_S = \&\{\text{enq} : \Pi x : A_S . \text{queue } A_S, \ \text{deq} : \oplus\{\text{none} : \text{queue } A_S, \ \text{some} : \exists x : A_S . \text{queue } A_S\}\}$
Example: shared queue

What should be the type of a shared queue?

\[
\begin{align*}
A_S & \triangleq \uparrow^S_L A_L \\
A_L, B_L & \triangleq \bigoplus \{ l : A_L \} | A_L \otimes B_L | 1 | \exists x : A_S. B_L | \\
& \quad \& \{ l : A_L \} | A_L \rightarrow B_L | \downarrow^S L A_S | \Pi x : A_S. B_L
\end{align*}
\]

queue \( A_S \) = \&\{ \text{enq} : \Pi x : A_S. \} \\
\quad \text{queue} \ A_S, \quad \text{deq} : \bigoplus \{ \text{none} : \} \\
\quad \text{queue} \ A_S, \quad \text{some} : \exists x : A_S. \quad \text{queue} \ A_S \} \}
Example: shared queue

What should be the type of a shared queue?

\[
\begin{align*}
A_S & \triangleq \uparrow^S_A L \\
A_L, B_L & \triangleq \oplus\{ l : A_L \} \mid A_L \otimes B_L \mid \mathbf{1} \mid \exists x : A_S. B_L \mid \\
& \quad \&\{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S L A_S \mid \forall x : A_S. B_L
\end{align*}
\]

queue \( A_S \) = \( \uparrow^S L \&\{ \text{enq} : \forall x : A_S. \text{queue} A_S, \text{deq} : \oplus\{ \text{none} : \text{queue} A_S, \text{some} : \exists x : A_S. \text{queue} A_S \} \} \)}
Example: shared queue

What should be the type of a shared queue?

\[ \text{queue } A_S = \uparrow^S_L \& \{ \text{enq : } \prod x : A_S. \downarrow^S_L \text{queue } A_S, \text{ deq : } \oplus \{ \text{none : } \downarrow^S_L \text{queue } A_S, \text{ some : } \exists x : A_S. \downarrow^S_L \text{queue } A_S \} \} \]
Example: shared queue

What should be the type of a shared queue?

\[
\begin{align*}
A_S & \triangleq \mathcal{L}\mathcal{S}_A \\
A_L, B_L & \triangleq \oplus\{l : A_L\} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S. B_L \\
& & \land\{l : A_L\} \mid A_L \rightarrow B_L \mid \mathcal{L}\mathcal{S}_A \mid \Pi x : A_S. B_L
\end{align*}
\]

Takeaway:
Example: shared queue

What should be the type of a shared queue?

\[
A_S \triangleq \uparrow^S_L A_L
\]

\[
A_L, B_L \triangleq \oplus \{ l : A_L \} \mid A_L \otimes B_L \mid \mathbf{1} \mid \exists x: A_S. B_L \mid \\
\& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S_L A_S \mid \Pi x: A_S. B_L
\]

\[
\text{queue } A_S = \uparrow^S_L \& \{ \text{enq} : \Pi x: A_S. \downarrow^S_L \text{queue } A_S, \\
\text{deq} : \oplus \{ \text{none} : \downarrow^S_L \text{queue } A_S, \text{some} : \exists x: A_S. \downarrow^S_L \text{queue } A_S \}\}
\]

Takeaway:

up-shift is an acquire
Example: shared queue

What should be the type of a shared queue?

\[
A_S \triangleq \uparrow_s A_L
\]

\[
A_L, B_L \triangleq \{l : A_L\} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S, B_L \mid
\&\{l : A_L\} \mid A_L \rightarrow B_L \mid \downarrow_s A_S \mid \Pi x : A_S, B_L
\]

queue \(A_S\) = \[\uparrow_s \&\{\text{enq} : \Pi x : A_S. \downarrow_s \text{queue } A_S, \text{deq} : \oplus\{\text{none} : \downarrow_s \text{queue } A_S, \text{some} : \exists x : A_S. \downarrow_s \text{queue } A_S\}\}\]

Takeaway:
- up-shift is an acquire
- down-shift is a release
Key idea 3: equi-synchronizing
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_s = \uparrow_L^S \& \{ \text{enq} : \prod x:A_s. \downarrow_L^S \text{ queue } A_s, \\
\text{deq} : \oplus \{ \text{none} : \downarrow_L^S \text{ queue } A_s, \text{some} : \exists x:A_s. \downarrow_L^S \text{ queue } A_s \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_s = \uparrow^s_L \land \{\text{enq} : \Pi x : A_s. \downarrow^s_L \text{ queue } A_s, \text{ deq} : \oplus \{\text{none} : \downarrow^s_L \text{ queue } A_s, \text{ some} : \exists x : A_s. \downarrow^s_L \text{ queue } A_s\}\}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^s_L \& \{ \text{enq} : \Pi x : A_S. \downarrow^s_L \text{queue } A_S, \\
\text{deq} : \oplus \{ \text{none} : \downarrow^s_L \text{queue } A_S, \text{some} : \exists x : A_S. \downarrow^s_L \text{queue } A_S \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^S \land \{\text{enq} : \Pi x : A_S. \downarrow^S \text{queue } A_S, \\
\text{deq} : \oplus \{\text{none} : \downarrow^S \text{queue } A_S, \text{some} : \exists x : A_S. \downarrow^S \text{queue } A_S\}\}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_s = \uparrow^s_l \& \{ \text{enq} : \Pi x : A_s. \downarrow^s_l \text{queue } A_s, \\
\text{deq} : \oplus \{ \text{none} : \downarrow^s_l \text{queue } A_s, \text{some} : \exists x : A_s. \downarrow^s_l \text{queue } A_s \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^L \& \{ \text{enq} : \Pi x : A_S. \downarrow^S \text{queue } A_S, \\
\text{deq} : \oplus \{ \text{none} : \downarrow^L \text{queue } A_S, \text{some} : \exists x : A_S. \downarrow^S \text{queue } A_S \} \}
\]

process is released back to same type previously acquired
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^S_L \& \{ \text{enq} : \Pi x : A_S. \downarrow^S_L \text{queue } A_S, \text{deq} : \oplus \{ \text{none} : \downarrow^S_L \text{queue } A_S, \text{some} : \exists x : A_S. \downarrow^S_L \text{queue } A_S \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_s = \uparrow^s_L \& \{ \text{enq : } \Pi x : A_s. \downarrow^s_L \text{ queue } A_s, \\
\text{deq : } \oplus \{ \text{none : } \downarrow^s_L \text{ queue } A_s, \text{some : } \exists x : A_s. \downarrow^s_L \text{ queue } A_s \} \}
\]

\[
\text{queue } A_s = \uparrow^s_L \& \{ \text{enq : } \Pi x : A_s. \downarrow^s_L \text{ queue } A_s, \\
\text{deq : } \oplus \{ \text{none : } \downarrow^s_L \uparrow^s_L 1, \text{some : } \exists x : A_s. \downarrow^s_L \text{ queue } A_s \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^S_L \& \{ \text{enq} : \Pi x : A_S. \downarrow^S_L \text{queue } A_S, \\
\quad \text{deq} : \oplus \{ \text{none} : \downarrow^S_L \text{queue } A_S, \text{some} : \exists x : A_S. \downarrow^S_L \text{queue } A_S \} \}
\]

\[
\text{queue } A_S = \uparrow^S_L \& \{ \text{enq} : \Pi x : A_S. \downarrow^S_L \text{queue } A_S, \\
\quad \text{deq} : \oplus \{ \text{none} : \downarrow^S_L \uparrow^S_L 1, \text{some} : \exists x : A_S. \downarrow^S_L \text{queue } A_S \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^S L \& \{ \text{enq} : \Pi x : A_S. \downarrow^S L \text{ queue } A_S, \\
\text{deq} : \oplus \{ \text{none} : \downarrow^S L \text{ queue } A_S, \text{some} : \exists x : A_S. \downarrow^S L \text{ queue } A_S \} \}
\]

\[
\text{queue } A_S = \uparrow^S L \& \{ \text{enq} : \Pi x : A_S. \downarrow^S L \text{ queue } A_S, \\
\text{deq} : \oplus \{ \text{none} : \downarrow^S L \uparrow^S L \text{ 1}, \text{some} : \exists x : A_S. \downarrow^S L \text{ queue } A_S \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^S_L \land \{ \text{enq : } \Pi x : A_S. \ \downarrow^S_L \text{ queue } A_S, \\
\text{deq : } \oplus \{ \text{none : } \downarrow^S_L \text{ queue } A_S, \ \text{some : } \exists x : A_S. \ \downarrow^S_L \text{ queue } A_S \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^S_L \& \{ \text{enq : } \Pi x : A_S. \ \downarrow^S_L \text{ queue } A_S, \\
\text{deq : } \oplus \{ \text{none : } \downarrow^S_L \text{ queue } A_S, \text{ some : } \exists x : A_S. \ \downarrow^S_L \text{ queue } A_S \} \}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

queue $A_s = \uparrow^s_L \& \{enq : \Pi x : A_s. \downarrow^s_L queue A_s,$

deq : $\oplus\{none : \downarrow^s_L queue A_s, \text{some} : \exists x : A_s. \downarrow^s_L queue A_s}\}$

process is released back to different type
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow_L^S \& \{ \text{enq} : \Pi x : A_S. \downarrow_L^S \text{ queue } A_S, \\
\text{deq} : \oplus \{ \text{none} : \downarrow_L^S \text{ queue } A_S, \text{some} : \exists x : A_S. \downarrow_L^S \text{ queue } A_S \} \}
\]

process is released back to different type

next client to acquire encounters protocol violation!
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_s = \uparrow\downarrow^S_{L} \& \{\text{enq} : \Pi x : A_s. \downarrow^S_{L} \text{ queue } A_s, \\
\text{deq} : \oplus \{\text{none} : \downarrow^S_{L} \text{ queue } A_s, \text{some} : \exists x : A_s. \downarrow^S_{L} \text{ queue } A_s\}\}
\]

\[
\text{queue } A_s = \uparrow\downarrow^S_{L} \& \{\text{enq} : \Pi x : A_s. \downarrow^S_{L} \text{ queue } A_s, \\
\text{deq} : \oplus \{\text{none} : \downarrow^S_{L} \uparrow^S_{L} 1, \text{some} : \exists x : A_s. \downarrow^S_{L} \text{ queue } A_s\}\}
\]
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow_L^S \& \{\text{enq} : \Pi x : A_S. \downarrow_L^S \text{ queue } A_S, \\
\text{deq} : \oplus \{\text{none} : \downarrow_L^S \text{ queue } A_S, \text{some} : \exists x : A_S. \downarrow_L^S \text{ queue } A_S\}\}
\]

\[
\text{queue } A_S = \uparrow_L^S \& \{\text{enq} : \Pi x : A_S. \downarrow_L^S \text{ queue } A_S, \\
\text{deq} : \oplus \{\text{none} : \downarrow_L \uparrow_L^S 1, \text{some} : \exists x : A_S. \downarrow_L^S \text{ queue } A_S\}\}
\]

equi-synchronizing: type wellformedness condition guaranteeing that any release is back to type at which previously acquired
Key idea 3: equi-synchronizing

Is mutual exclusion enough for restoring preservation?

\[
\text{queue } A_S = \uparrow^S \& \{\text{enq} : \Pi x : A_S. \ \downarrow^L \text{queue } A_S, \\
\text{deq} : \oplus \{\text{none} : \downarrow^L \text{queue } A_S, \ \text{some} : \exists x : A_S. \ \downarrow^S \text{queue } A_S\}\}
\]

\[
\text{queue } A_S = \uparrow^S \& \{\text{enq} : \Pi x : A_S. \ \downarrow^L \text{queue } A_S, \\
\text{deq} : \oplus \{\text{none} : \downarrow^L \uparrow^S 1, \ \text{some} : \exists x : A_S. \ \downarrow^L \text{queue } A_S\}\}
\]

equi-synchronizing: type wellformedness condition guaranteeing that any release is back to type at which previously acquired

acquire-release and equi-synchronizing guarantee preservation
Typing judgments
Typing judgments

<table>
<thead>
<tr>
<th>weakening</th>
<th>contraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>↑</td>
</tr>
</tbody>
</table>

| $A_S$ | $\triangleq$ | $\uparrow^S_L A_L$ |
| $A_L, B_L$ | $\triangleq$ | $\oplus \{ l : A_L \} | A_L \otimes B_L | 1 | \exists x : A_S . B_L |$
|         |             | $\& \{ l : A_L \} | A_L \to B_L | \downarrow^S_L A_S | \Pi x : A_S . B_L$ |
Typing judgments

\[ A_S \triangleq ^S_L A_L \]

\[ A_L, B_L \triangleq \oplus \{ l : \overline{A_L} \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S . B_L \mid \& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S . B_L \]

\[ \Gamma \vdash \Sigma \ P :: (x_S : A_S) \]  
shared process \( P \), providing session of type \( A_S \) along \( x_S \), using channels in \( \Gamma \)

\[ \Gamma ; \Delta \vdash \Sigma \ P :: (x_L : A_L) \]  
linear process \( P \), providing session of type \( A_L \) along \( x_L \), using channels in \( \Gamma \) and \( \Delta \)

\[ \Gamma \]  
shared (structural) context

\[ \Delta \]  
linear context
Typing judgments

<table>
<thead>
<tr>
<th>Weakening</th>
<th>Contraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_S \triangleq \uparrow^S A_L$</td>
<td></td>
</tr>
<tr>
<td>$A_L, B_L \triangleq \oplus {l : A_L} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S, B_L \mid$</td>
<td></td>
</tr>
<tr>
<td>$&amp; {l : A_L} \mid A_L \rightarrow B_L \mid \downarrow^S A_S \mid \Pi x : A_S \cdot B_L$</td>
<td></td>
</tr>
</tbody>
</table>

$\Gamma \vdash \Sigma \ P :: (x_S : A_S)$  
shared process $P$, providing session of type $A_S$ along $x_S$, using channels in $\Gamma$

$\Gamma; \Delta \vdash \Sigma \ P :: (x_L : A_L)$  
linear process $P$, providing session of type $A_L$ along $x_L$, using channels in $\Gamma$ and $\Delta$

$\Gamma$  
shared (structural) context

$\Delta$  
linear context
Typing judgments

\( A_S \triangleq \uparrow^S_L A_L \)

\[
A_L, B_L \triangleq \oplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S. B_L \\
\& \{ l : A_L \} \mid A_L \circ B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S. B_L
\]

\( \Gamma \vdash \Sigma \ P :: (x_S : A_S) \)  
shared process \( P \), providing session of type \( A_S \)  
along \( x_S \), using channels in \( \Gamma \)

\( \Gamma; \triangleq \Delta \vdash \Sigma \ P :: (x_L : A_L) \)  
linear process \( P \), providing session of type \( A_L \)  
along \( x_L \), using channels in \( \Gamma \) and \( \Delta \)

\( \Gamma \)  
shared (structural) context

\( \Delta \)  
linear context
Typing judgments

Weakening contraction

\[ A_S \triangleq \uparrow^S A_L \]

\[ A_L, B_L \triangleq \oplus \{ l : A_L \} | A_L \otimes B_L | 1 | \exists x : A_S . B_L | \]
\[ \& \{ l : A_L \} | A_L \rightarrow B_L | \downarrow^S A_S | \Pi x : A_S . B_L \]

\[ \Gamma \vdash_{\Sigma} P :: (x_S : A_S) \] shared process \( P \), providing session of type \( A_S \)
along \( x_S \), using channels in \( \Gamma \)

\[ \Gamma ; \Delta \vdash_{\Sigma} P :: (x_L : A_L) \] linear process \( P \), providing session of type \( A_L \)
along \( x_L \), using channels in \( \Gamma \) and \( \Delta \)

\[ \Gamma \] shared (structural) context

\[ \Delta \] linear context
Typing judgments

\[ A_S \triangleq \uparrow^S_L A_L \]
\[ A_L, B_L \triangleq \oplus \{ l : A_L \} | A_L \otimes B_L | 1 | \exists x : A_S. B_L | \]
\[ \& \{ l : A_L \} | A_L \rightarrow B_L | \downarrow^S_L A_S | \Pi x : A_S. B_L \]

\[ \Gamma \vdash \Sigma \quad P :: (x_S : A_S) \quad \text{shared process } P, \text{ providing session of type } A_S \]
\[ \text{along } x_S, \text{ using channels in } \Gamma \]

\[ \Gamma ; \Delta \vdash \Sigma \quad P :: (x_L : A_L) \quad \text{linear process } P, \text{ providing session of type } A_L \]
\[ \text{along } x_L, \text{ using channels in } \Gamma \text{ and } \Delta \]

\[ \Gamma \quad \text{shared (structural) context} \]

\[ \Delta \quad \text{linear context} \]
Typing judgments

\[ A_S \triangleq \uparrow^S_A L \]

\[ A_L, B_L \triangleq \bigoplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S , B_L \mid \]

\[ \& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S_A S \mid \Pi x : A_S \cdot B_L \]

\[ \Gamma \vdash \Sigma \ P :: (x_S : A_S) \quad \text{shared process } P, \text{ providing session of type } A_S \text{ along } x_S, \text{ using channels in } \Gamma \]

\[ \Gamma ; \Delta \vdash \Sigma \ P :: (x_L : A_L) \quad \text{linear process } P, \text{ providing session of type } A_L \text{ along } x_L, \text{ using channels in } \Gamma \text{ and } \Delta \]

\[ \Gamma \quad \text{shared (structural) context} \]

\[ \Delta \quad \text{linear context} \]
Acquire
Acquire

\[
\frac{\Gamma, x_s : \uparrow^s_L A_L; \ \Delta, x_L : A_L \vdash \Sigma Q_{x_L} :: (z_L : C_L)}{\Gamma, x_s : \uparrow^s_L A_L; \ \Delta \vdash \Sigma x_L \leftarrow \text{acquire} \ x_s ; Q_{x_L} :: (z_L : C_L)} \quad (T-\uparrow^s_{LL})
\]
Acquire

\[
\frac{\Gamma, x_s : \uparrow^s A \quad \Delta, x_L : A \vdash Q \cdot x_L :: (z_L : C_L)}{\Gamma, x_s : \uparrow^s A \quad \Delta \vdash x_L \leftarrow \text{acquire} \ x_s \ ; \ Q \cdot x_L :: (z_L : C_L) \quad (T-\uparrow^s_{LL})}
\]
Acquire

\[
\Gamma, x_S : \uparrow^s L A_L; \quad \Delta, x_L : A_L \vdash_{\Sigma} Q_{x_L} :: (z_L : C_L) \\
\Gamma, x_S : \uparrow^s L A_L; \quad \Delta \vdash_{\Sigma} x_L \leftarrow \text{acquire} \; x_S \; ; Q_{x_L} :: (z_L : C_L) \\
\Gamma; \; \vdash_{\Sigma} P_{x_L} :: (x_L : A_L) \\
\Gamma \vdash_{\Sigma} x_L \leftarrow \text{accept} \; x_S \; ; P_{x_L} :: (x_S : \uparrow^s L A_L) \quad (T-\uparrow^s_{LL})
\]

\[
\Gamma; \; \vdash_{\Sigma} P_{x_L} :: (x_L : A_L) \\
\Gamma \vdash_{\Sigma} x_L \leftarrow \text{accept} \; x_S \; ; P_{x_L} :: (x_S : \uparrow^s L A_L) \quad (T-\uparrow^s_{LR})
\]
Acquire

\[
\Gamma, x_S : \uparrow^s L A_L; \Delta, x_L : A_L \vdash \Sigma Q_{x_L} :: (z_L : C_L)
\]

\[
\Gamma, x_S : \uparrow^s L A_L; \Delta \vdash \Sigma x_L \leftarrow \text{acquire } x_S ; Q_{x_L} :: (z_L : C_L)
\]

\[
\Gamma; \cdot \vdash \Sigma P_{x_L} :: (x_L : A_L)
\]

\[
\Gamma \vdash \Sigma x_L \leftarrow \text{accept } x_S ; P_{x_L} :: (x_S : \uparrow^s L A_L)
\]
Acquire

\[
\Gamma, x_S : \uparrow^s_L A_L; \; \Delta, x_L : A_L \vdash \Sigma \; Q_{x_L} :: (z_L : C_L) \\
\Gamma, x_S : \uparrow^s_L A_L; \; \Delta \vdash \Sigma \; x_L \leftarrow \text{acquire} \; x_S ; \; Q_{x_L} :: (z_L : C_L)
\]

\[
\Gamma; \cdot \vdash \Sigma \; P_{x_L} :: (x_L : A_L) \\
\Gamma \vdash \Sigma \; x_L \leftarrow \text{accept} \; x_S ; \; P_{x_L} :: (x_S : \uparrow^s_L A_L)
\]

\[
(D-\uparrow^s_L) \quad \text{proc}(a_S, x_L \leftarrow \text{accept} \; a_S ; \; P_{x_L}), \; \text{proc}(c_L, x_L \leftarrow \text{acquire} \; a_S ; \; Q_{x_L}) \\
\quad \rightarrow \; \text{unvail}(a_S), \; \text{proc}(a_L, [a_L/x_L] \; P_{x_L}), \; \text{proc}(c_L, [a_L/x_L] \; Q_{x_L})
\]
Release
Release

\[ \Gamma, x_S : A_S; \Delta \vdash x_S \trianglerighteq (z_L : C_L) \]

\[ \frac{\Gamma; \Delta, x_L : \downarrow_S A_S \vdash x_S \leftarrow \text{release } x_L ; Q_{x_S} :: (z_L : C_L)}{} \quad (T-\downarrow^S_{LL}) \]
Release

\[
\frac{\Gamma, x_S : A_S; \Delta \vdash_\Sigma Q x_S :: (z_L : C_L)}{
\Gamma; \Delta, x_L : \downarrow_l^S A_S \vdash_\Sigma x_S \leftarrow \text{release } x_L ; Q x_S :: (z_L : C_L)}
\]  

(T-$\downarrow_l^S$L)
Release

\[
\Gamma, x_S : A_S; \quad \Delta \vdash \Sigma Q_{xs} :: (z_L : C_L)\\
\frac{\Gamma; \quad \Delta, x_L : \downarrow^S A_S \vdash \Sigma x_S \leftarrow \text{release } x_L ; Q_{xs} :: (z_L : C_L)}{\Gamma; \quad \Sigma P_{xs} :: (x_S : A_S) (T-\downarrow^S_{L_L})}\\
\frac{\Gamma; \quad \cdot \vdash \Sigma x_S \leftarrow \text{detach } x_L ; P_{xs} :: (x_L : \downarrow^S A_S)}{\Gamma; \quad \cdot \vdash \Sigma x_S \leftarrow \text{detach } x_L ; P_{xs} :: (x_L : \downarrow^S A_S) (T-\downarrow^S_{L_R})}
\]
Release

\[ \Gamma, x_S : A_S; \quad \Delta \vdash \Sigma Q_{x_S} :: (\triangleright L : C_L) \]

\[ \Gamma; \quad \Delta, x_L : \llbracket L \rrbracket A_S \vdash \Sigma x_S \leftarrow \text{release} \ x_L ; Q_{x_S} :: (\triangleright L : C_L) \]  \hspace{1cm} (T-\triangleright L_L)

\[ \Gamma \vdash \Sigma P_{x_S} :: (x_S : A_S) \]

\[ \Gamma; \quad \cdot \vdash \Sigma x_S \leftarrow \text{detach} \ x_L ; P_{x_S} :: (x_L : \llbracket L \rrbracket A_S) \]  \hspace{1cm} (T-\triangleright L_R)
Release

\[
\frac{\Gamma, x_S : A_S; \Delta \vdash \sum Q_{x_S} :: (z_L : C_L)}{\Gamma; \Delta, x_L : \downarrow^s_L A_S \vdash \sum x_S \leftarrow \text{release } x_L ; Q_{x_S} :: (z_L : C_L)} \quad (T-\downarrow^s_L L)
\]

\[
\frac{\Gamma \vdash \sum P_{x_S} :: (x_S : A_S)}{\Gamma; \cdot \vdash \sum x_S \leftarrow \text{detach } x_L ; P_{x_S} :: (x_L : \downarrow^s_L A_S)} \quad (T-\downarrow^s_L R)
\]

\[
(D-\downarrow^s_L) \quad \text{proc}(a_L, x_S \leftarrow \text{detach } a_L ; P_{x_S}), \text{proc}(c_L, x_S \leftarrow \text{release } a_L ; Q_{x_S}), \text{unvail}(a_S)
\]

\[
\rightarrow \quad \text{proc}(a_S, [a_S/x_S] P_{x_S}), \text{proc}(c_L, [a_S/x_S] Q_{x_S})
\]
Let’s implement a shared queue in SILLs
Taking stock
Taking stock

We have a session type system that allows shared and linear channels to coexist and guarantees:
Taking stock

We have a session type system that allows shared and linear channels to coexist and guarantees:

data-race-freedom (low-level and high-level)
Taking stock

We have a session type system that allows shared and linear channels to coexist and guarantees:

- Data-race-freedom (low-level and high-level)
- Protocol adherence
Taking stock

We have a session type system that allows shared and linear channels to coexist and guarantees:

- data-race-freedom (low-level and high-level)
- protocol adherence

What about deadlock-freedom?
Taking stock

We have a session type system that allows shared and linear channels to coexist and guarantees:

- data-race-freedom (low-level and high-level)
- protocol adherence

What about deadlock-freedom?

- unfortunately we have lost deadlock-freedom
Taking stock

We have a session type system that allows shared and linear channels to coexist and guarantees:

- data-race-freedom (low-level and high-level)
- protocol adherence

What about deadlock-freedom?

unfortunately we have lost deadlock-freedom