Session-Typed Concurrent Programming
Lecture 1

Stephanie Balzer
Carnegie Mellon University

OPLSS 2021
June 23, 2021
About this class
About this class

Session-type concurrent programming

- concurrency (as opposed to parallelism)
- nondeterminism
About this class

Session-type concurrent programming

- concurrency (as opposed to parallelism)
- nondeterminism

Roadmap

- message-passing concurrent programming
- session types as types for message-passing concurrency
- linear logic and session types
- manifest sharing (controlled form of aliasing)
- deadlock-freedom
Terminology that will be meaningful to you
Terminology that will be meaningful to you

- session type
- linear logic
- sequent calculus
- weakening
- contraction
- aliasing
- cut
- pi-calculus
- higher-order channels
- preservation
- identity
- progression
- Curry-Howard correspondence
- deadlock-freedom
- session fidelity
- affine
- intuitionism
Learning objectives
Learning objectives

• How to program in a message-passing, concurrent style
Learning objectives

- How to program in a message-passing, concurrent style
- What session types are about
Learning objectives

• How to program in a message-passing, concurrent style
• What session types are about
• Benefits of linear logic for programming
Learning objectives

• How to program in a message-passing, concurrent style
• What session types are about
• Benefits of linear logic for programming
• How to accommodate sharing in a logically motivated way
Learning objectives

• How to program in a message-passing, concurrent style
• What session types are about
• Benefits of linear logic for programming
• How to accommodate sharing in a logically motivated way
• How to reason about deadlocks in the presence of aliasing
Hands-on session
Hands-on session

Tutorial by Soares Chen (Ruo Fei)

- Friday (6/25) and Saturday (6/26) from 12:20 pm - 1:50 pm
Hands-on session

Tutorial by Soares Chen (Ruo Fei)
• Friday (6/25) and Saturday (6/26) from 12:20 pm - 1:50 pm

Ferrite session type library in Rust
• writing session-typed programs in Rust
• support of linear and shared session types
Hands-on session

Tutorial by Soares Chen (Ruo Fei)

- Friday (6/25) and Saturday (6/26) from 12:20 pm - 1:50 pm

Ferrite session type library in Rust

- writing session-typed programs in Rust
- support of linear and shared session types

What you’ll learn

- techniques used for session types embedding
- how to use the library
- practice with prepared exercises
Message-passing concurrent programming
Message-passing programming model
Message-passing programming model

Computation by a processes that exchange messages along channels
Message-passing programming model

Computation by processes that exchange messages along channels

Legend:  process
Message-passing programming model

Computation by a processes that exchange messages along channels

Legend: process  channel
Message-passing programming model

Computation by a processes that exchange messages along channels
Message-passing programming model

Computation by processes that exchange messages along channels

Legend:  
- gray circle: process  
- black line: channel

n-ary channels: c shared among P3, P4, and P5
Message-passing programming model

Computation by processes that exchange messages along channels

**Legend:**
- □ process
- channel

n-ary channels: channel c is shared among P3, P4, and P5

nondeterminism: if P3 sends a message along channel c, either P4 or P5 can get it.
Message-passing programming model

Computation by a processes that exchange messages along channels
Message-passing programming model

Computation by a processes that exchange messages along channels

underlying formal model: process calculus (e.g., pi-calculus)

Message-passing programming model

Computation by processes that exchange messages along channels

- underlying formal model: process calculus (e.g., pi-calculus)
- universality: encoding of lambda-calculus into pi-calculus


A message-passing queue
A message-passing queue

Queue of character processes:
A message-passing queue

Queue of character processes:

```
 dequeue
```

```
“O” — “P” — “L” — “S”
```
A message-passing queue

Queue of character processes:
A message-passing queue

Queue of character processes:
A message-passing queue

Queue of character processes:
A message-passing queue

Queue of character processes:

client → enq “S” → “O” → “P” → “L” → “S”
A message-passing queue

Queue of character processes:

client → enq “S” → “O” → “P” → “L” → “S” → “S” → new process
A message-passing queue

Queue of character processes:

client

“O” “P” “L” “S” “S”
A message-passing queue

Queue of character processes:
A message-passing queue

Queue of character processes:

client deq "O" "P" "L" "S" "S"
A message-passing queue

Queue of character processes:

client

“P” “L” “S” “S”
A message-passing queue
A message-passing queue

Here, we’ve exchanged basic values (e.g., characters).
A message-passing queue

Here, we’ve exchanged basic values (e.g., characters).

In original pi-calculus, only channel references can be exchanged.
A message-passing queue

Here, we’ve exchanged basic values (e.g., characters).

In original pi-calculus, only channel references can be exchanged.
A message-passing queue

Here, we’ve exchanged basic values (e.g., characters).

In original pi-calculus, only channel references can be exchanged.
A message-passing queue

Here, we’ve exchanged basic values (e.g., characters).

In original pi-calculus, only channel references can be exchanged.

“mobility” in pi-calculus
A message-passing queue

Here, we’ve exchanged basic values (e.g., characters).

In original pi-calculus, only channel references can be exchanged.

“mobility” in pi-calculus

“higher-order channels” in session types
Session types
Types for protocols of message exchange
Types for protocols of message exchange

Session types

Types for protocols of message exchange

Session types

\[ A \triangleq \ ?[T].A' \ | \ ![T].A' \ | \&\{l_1 : A_1, \ldots, l_n : A_n\} \ | \ \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \ | \ \text{end} \ | \ X \ | \ \mu X.A' \]

\[ T \triangleq \ A \ | \ \text{int} \ | \ \ldots \]
Session types

\[
\begin{align*}
A & \triangleq \quad ?[T].A' \mid ![T].A' \mid \\
& \quad \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
& \quad \text{end} \mid X \mid \mu X.A' \\
T & \triangleq \quad A \mid \text{int} \mid \ldots
\end{align*}
\]

input: receive message of type \(T\), continue as type \(A'\)
Types for protocols of message exchange

Session types

\[ A \triangleq \begin{array}{c}
?T.A' \\
\&\{l_1 : A_1, \ldots, l_n : A_n\} \\
\oplus\{l_1 : A_1, \ldots, l_n : A_n\} \\
\text{end} \\
X \\
\mu X.A'
\end{array} \]

\[ T \triangleq A | \text{int} | \ldots \]

input: receive message of type T, continue as type A'

types can be session (higher-order channels) or basic types

Types for protocols of message exchange

Session types

\[ A \triangleq \ ?[T].A' \ | \ ![T].A' \ | \ \&\{l_1 : A_1, \ldots, l_n : A_n\} \ | \ \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \ | \ \text{end} \ | \ X \ | \ \mu X.A' \]

\[ T \triangleq A \ | \ \text{int} \ | \ \ldots \]
Types for protocols of message exchange

Session types

\[ A \triangleq \begin{array}{c}
?\![T].A' \\
\&\{l_1 : A_1, \ldots, l_n : A_n\} \\
\oplus\{l_1 : A_1, \ldots, l_n : A_n\} \\
\text{end} \\
X \\
\mu X.A'
\end{array} \]

\[ T \triangleq \begin{array}{c}
A \\
\text{int} \\
\ldots
\end{array} \]

output: send message of type T, continue as type A'
Types for protocols of message exchange

Session types

\[ A \triangleq \begin{array}{c}
?T].A' \mid !T].A' \mid \\
\&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
\text{end} \mid X \mid \mu X.A' \\
\end{array} \]

\[ T \triangleq A \mid \text{int} \mid \ldots \]
Types for protocols of message exchange

Session types

\[ A \triangleq ?[T].A' \mid ![T].A' \mid \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \text{end} \mid X \mid \mu X . A' \]

\[ T \triangleq A \mid \text{int} \mid \ldots \]

external choice: receive label \( l_i \), continue as type \( A_i \)
Types for protocols of message exchange

Session types

\[ A \triangleq ?[T].A' \mid ![T].A' \mid \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \text{end} \mid X \mid \mu X.A' \]

\[ T \triangleq A \mid \text{int} \mid \ldots \]
Types for protocols of message exchange

Session types

\[ A \equiv \ ?[T].A' \ | \ ![T].A' \ | \ \&\{l_1 : A_1, \ldots, l_n : A_n\} \ | \ \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \ | \ \text{end} \ | \ A \ | \ \mu X.A' \]

\[ T \equiv \ A \ | \ \text{int} \ | \ \ldots \]

internal choice: send label \(l_i\), continue as type \(A_i\)

Types for protocols of message exchange

Session types

\[ A \triangleq \ ?[T].A' \ | \ ![T].A' \ | \ \&\{l_1 : A_1, \ldots, l_n : A_n\} \ | \ \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \ | \ \text{end} \ | \ X \ | \ \mu X.A' \]

\[ T \triangleq \ A \ | \ \text{int} \ | \ \ldots \]
Types for protocols of message exchange

Session types

\[
A \triangleq \quad ?[T].A' \mid ![T].A' \mid \\
\&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
\text{end} \mid X \mid \mu X.A'
\]

\[
T \triangleq \quad A \mid \text{int} \mid \ldots
\]

termination: close session and terminate

Types for protocols of message exchange

Session types

\[ A \triangleq \square[T].A' \mid \lozenge[T].A' \mid \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \text{end} \mid X \mid \mu X.A' \]

\[ T \triangleq A \mid \text{int} \mid \ldots \]
Types for protocols of message exchange

Session types

\[ A \triangleq ?[T].A' \mid ![T].A' \mid \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \text{end} \mid X \mid \mu X.A' \]

\[ T \triangleq A \mid \text{int} \mid \ldots \]

recursive session types

Types for protocols of message exchange

Session types

\[ A \triangleq \ ?[T].A' \mid ![T].A' \mid \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \text{end} \mid X \mid \mu X.A' \]

\[ T \triangleq A \mid \text{int} \mid \ldots \]
Types for protocols of message exchange

Session types

\[ A \triangleq ?[T].A' \mid ![T].A' \mid \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \text{end} \mid X \mid \mu X.A' \]

\[ T \triangleq A \mid \text{int} \mid \ldots \]

Queue session type:
Types for protocols of message exchange

Session types

\[ A \triangleq [?T].A' \mid ![T].A' \mid \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \text{end} \mid X \mid \mu X.A' \]

\[ T \triangleq A \mid \text{int} \mid \ldots \]

Queue session type:

\[
\text{queue} = \&\{\text{enq} : [?\text{char}].\text{queue}, \\
\text{deq} : \oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}\}
\]
Queue session type in action

Queue session type:

```plaintext
queue = &{enq : ?[char].queue,
    deq : ⊕{none : end, some : ![char].queue}}
```
Queue session type in action

Queue session type:

\[
\text{queue} = \&\{\text{enq} : ?[\text{char}].\text{queue}, \\
\text{deq} : \oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}\}
\]
Queue session type in action

Queue session type:

```
queue = \{enq : ?[char].queue,
      deq : \{none : end, some : ![char].queue\}\}
```

Type of channel q:
Queue session type in action

Queue session type:

```plaintext
queue = &\{enq : ?[char].queue,
    deq : \{none : end, some : ![char].queue\}\}
```

Type of channel q: queue
Queue session type in action

Queue session type:

```latex
\text{queue} = \&\{\text{enq : } ?[\text{char}].\text{queue}, \\
\text{deq : } \oplus\{\text{none : end, some : !}[\text{char}].\text{queue}\}\}
```

Type of channel q: queue
Queue session type in action

Queue session type:

\[
\text{queue} = \&\{\text{enq : ?[char].queue}, \\
\text{deq : } \oplus\{\text{none : end, some : ![char].queue}\}\}
\]

Type of channel \( q \): \?[char].queue
Queue session type in action

Queue session type:

\[
\text{queue} = \&\{\text{enq : ?[char].queue,} \\
\text{deq : } \oplus\{\text{none : end, some : !}[\text{char}.\text{queue}\}}\}
\]

Type of channel q: \([\text{char}].\text{queue}\)
Queue session type in action

Queue session type:

```plaintext
queue = &{enq : ?[char].queue,
    deq : ⊕{none : end, some : ![char].queue}}
```

Type of channel q:  queue
Queue session type in action

Queue session type:

\[
\begin{align*}
\text{queue} &= \&\{\text{enq} : \?\text{[char].queue}, \\
\text{deq} : \oplus\{\text{none} : \text{end}, \text{some} : !\text{[char].queue}\}\}
\end{align*}
\]

Type of channel \( q \): queue
Queue session type in action

Queue session type:

\[
\text{queue} = \& \{ \text{enq} : ?[\text{char}].\text{queue}, \\
\text{deq} : \oplus \{ \text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue} \} \}
\]

Type of channel q: \(\oplus \{ \text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue} \}\)
Queue session type in action

Queue session type:

\[
\text{queue} = \&\{\text{enq} : ?[\text{char}].\text{queue}, \\
\text{deq} : \oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}\}
\]

Type of channel q:

\[
\oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}
\]
Queue session type in action

Queue session type:

```
queue = &{
    enq : ?[char].queue,
    deq : ⊕{none : end, some : ![char].queue}
}
```

Type of channel q: ![char].queue
Queue session type in action

Queue session type:

\[
\text{queue} = \&\{\text{enq} : \?\text{[char].queue,}
\text{deq} : \oplus\{\text{none : end, some : !\text{[char].queue}}\}\}
\]

Type of channel q: \text{!\text{[char].queue}}

client \hspace{2cm} q \hspace{2cm} \text{queue}
Queue session type in action

Queue session type:

\[
\text{queue} = \& \{\text{enq} : ?[\text{char}].\text{queue},
\quad \text{deq} : \oplus \{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}\}
\]

Type of channel \(q\): \text{queue}
Queue session type in action

Queue session type:

\[
\text{queue} = \&\{\text{enq : }?[\text{char}].\text{queue}, \\
\text{deq : }\oplus\{\text{none : end, some : !}[\text{char}].\text{queue}\}\}
\]

Type of channel q: queue

type of channel/process changes with message exchange
Protocol verification
Protocol verification

session types ensure protocol adherence by type-checking
Protocol verification

- Session types ensure protocol adherence by type-checking
- Session fidelity (a.k.a. preservation)
Protocol verification

- Session types ensure protocol adherence by type-checking
- Session fidelity (a.k.a. preservation)

Challenges for preservation:
Protocol verification

Session types ensure protocol adherence by type-checking

Session fidelity (a.k.a. preservation)

Challenges for preservation:

queue
Protocol verification

- Session types ensure protocol adherence by type-checking.
- Session fidelity (a.k.a. preservation)

Challenges for preservation:

```
client 1 → queue
  q → “O” → “P” → “L” → “S”
```
Protocol verification

- Session types ensure protocol adherence by type-checking.
- Session fidelity (a.k.a. preservation).

Challenges for preservation:

![Diagram showing client 1 and client 2 connected to a queue with the characters "O", "P", "L", "S" indicating preservation.]
Protocol verification

Session types ensure protocol adherence by type-checking.

Session fidelity (a.k.a. preservation)

Challenges for preservation:

Client 1

Client 2

Queue q

Queue "O" "P" "L" "S"

q: queue
Protocol verification

- Session types ensure protocol adherence by type-checking
- Session fidelity (a.k.a. preservation)

Challenges for preservation:

client 1

client 2

q: queue
Protocol verification

- Session types ensure protocol adherence by type-checking.
- Session fidelity (a.k.a. preservation).

Challenges for preservation:

```
client 1

q

client 2

queue

“O” —> “P” —> “L” —> “S”

q: ?[char].queue
```
Protocol verification

- Session types ensure protocol adherence by type-checking.
- Session fidelity (a.k.a. preservation).

Challenges for preservation:

```
client 1
q
“O” “P” “L” “S”
client 2
deq
```

q: `?[char].queue`
Protocol verification

- Session types ensure protocol adherence by type-checking
- Session fidelity (a.k.a. preservation)

Challenges for preservation:

Client 1

Client 2

Queue

Protocol violation

q: \(?[\text{char}].\text{queue}\)
Protocol verification
Protocol verification

Preservation: expectation for type of client and provider match

client 1

client 2

q

“O” “P” “L” “S”

queue
Protocol verification

Preservation: expectation for type of client and provider match

Strategies for recovery:
Protocol verification

Preservation: expectation for type of client and provider match

Strategies for recovery:

employ linearity/ownership to restrict to single client
Protocol verification

Preservation: expectation for type of client and provider match

Strategies for recovery:

- employ linearity/ownership to restrict to single client
- disallow multiple clients
Protocol verification

Preservation: expectation for type of client and provider match

Strategies for recovery:

- employ linearity/ownership to restrict to single client
- disallow multiple clients
- allow multiple clients but control aliasing (manifest sharing)
Intuitionistic linear logic session types
Linear logic from a programming perspective
Linear logic from a programming perspective

Linear logic is a so-called substructural logic that tracks ownership.
Linear logic from a programming perspective

Linear logic is a so-called substructural logic that tracks ownership. We first discover characteristics programmatically, then revisit them formally.
Linear logic from a programming perspective

Linear logic is a so-called substructural logic that tracks ownership. We first discover characteristics programmatically, then revisit them formally.

Types:

\[
\begin{align*}
A, B & \triangleq A \otimes B \quad \text{multiplicative conjunction} & \quad \text{“channel output”} \\
      & \quad A \to B \quad \text{multiplicative implication} & \quad \text{“channel input”} \\
      & \quad A \& B \quad \text{additive conjunction} & \quad \text{“external choice”} \\
      & \quad A \oplus B \quad \text{additive disjunction} & \quad \text{“internal choice”} \\
      & \quad 1 \quad \text{unit for } \otimes & \quad \text{“termination”}
\end{align*}
\]
Linear logic from a programming perspective

Linear logic is a so-called substructural logic that tracks ownership.

We first discover characteristics programmatically, then revisit them formally.

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”

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

For simplicity, we restrict to binary external/internal choice and to higher-order channels.
Linear logic from a programming perspective

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”
$1$  unit for $\otimes$  “termination”
Linear logic from a programming perspective

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”

Queue session type:
Linear logic from a programming perspective

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”

Queue session type:

\[
\text{queue } A = \& \{ \text{enq} : A \multimap \text{queue } A, \\
\text{deq} : \oplus \{ \text{none} : 1, \text{some} : A \otimes \text{queue } A \} \} 
\]
Linear logic from a programming perspective

Types:

\[ A, B \triangleq A \otimes B \quad \text{multiplicative conjunction} \]
\[ A \multimap B \quad \text{multiplicative implication} \]
\[ A \& B \quad \text{additive conjunction} \]
\[ A \oplus B \quad \text{additive disjunction} \]
\[ 1 \quad \text{unit for } \otimes \]

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

Queue session type: polymorphic

\[
\begin{align*}
\text{queue } A &= \& \{ \text{enq : } A \multimap \text{queue } A, \\
\text{deq : } \oplus \{ \text{none : 1, some : } A \otimes \text{queue } A \} \}
\end{align*}
\]
Typing judgment and rules
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process P offers a session of type A along channel x using session A_1, ..., A_n provided along channels x_1, ..., x_n.”
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process P offers a session of type A along channel x using session A_1, ..., A_n provided along channels x_1, ..., x_n.”
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process P offers a session of type A along channel x using session \( A_1, \ldots, A_n \) provided along channels \( x_1, \ldots, x_n \).”
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process P offers a session of type A along channel x using session A_1, ..., A_n provided along channels x_1, ..., x_n.”

Inference rule:

\[
\frac{
\Delta' \vdash Q :: (x : A')
}{
\Delta \vdash P; Q :: (x : A)
}\]
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process \( P \) offers a session of type \( A \) along channel \( x \) using session \( A_1, \ldots, A_n \) provided along channels \( x_1, \ldots, x_n \).”

Inference rule:

\[
\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A) \text{ conclusion}}
\]
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process P offers a session of type A along channel x using session A_1, ..., A_n provided along channels x_1, ..., x_n.”

Inference rule:

\[
\begin{array}{c}
\text{premise} \\
\Delta' \vdash Q :: (x : A')
\end{array}
\]

\[
\begin{array}{c}
\text{premise} \\
\Delta \vdash P; Q :: (x : A)
\end{array}
\]
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process P offers a session of type A along channel x using session \( A_1, \ldots, A_n \) provided along channels \( x_1, \ldots, x_n \).”

Inference rule:

\[
\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}
\]

bottom-up reading
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process \( P \) offers a session of type \( A \) along channel \( x \) using session \( A_1, \ldots, A_n \) provided along channels \( x_1, \ldots, x_n \).”

Inference rule:

\[
\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}
\]

bottom-up reading

premise conclusion current
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

"Process P offers a session of type A along channel x using session A_1, ..., A_n provided along channels x_1, ..., x_n."

Inference rule:

\[
\frac{\Delta ' \vdash Q :: (x : A')}{
\Delta \vdash P; Q :: (x : A)}
\]
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

"Process \( P \) offers a session of type \( A \) along channel \( x \) using session \( A_1, \ldots, A_n \) provided along channels \( x_1, \ldots, x_n \)."

Inference rule:

\[
\begin{align*}
\text{premise} & \quad \Delta' \vdash Q :: (x : A') \\
\text{conclusion} & \quad \Delta \vdash P; Q :: (x : A)
\end{align*}
\]

bottom-up reading

Left and right rules:

\[
\begin{align*}
\Delta', x : B & \vdash Q :: (z : C) \\
\Delta, x : A \triangleright B & \vdash P; Q :: (z : C) \quad \diamond_L \\
\Delta' & \vdash Q :: (x : B) \\
\Delta & \vdash P; Q :: (x : A \triangleright B) \quad \diamond_R
\end{align*}
\]
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process P offers a session of type A along channel x using session \( A_1, \ldots, A_n \) provided along channels \( x_1, \ldots, x_n \).”

Inference rule:

\[
\begin{align*}
\text{premise} & & \Delta' \vdash Q :: (x : A') \\
\text{conclusion} & & \Delta \vdash P; Q :: (x : A)
\end{align*}
\]

Left and right rules:

\[
\begin{align*}
\Delta', x : B & \vdash Q :: (z : C) \\
\Delta, x : A \odot B & \vdash P; Q :: (z : C)
\end{align*}
\] \( \diamond_L \)

\[
\begin{align*}
\Delta' & \vdash Q :: (x : B) \\
\Delta & \vdash P; Q :: (x : A \odot B)
\end{align*}
\] \( \diamond_R \)
Typing judgment and rules

Intuitionistic linear sequent:

\[ x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A) \]

“Process P offers a session of type A along channel x using session A_1, ..., A_n provided along channels x_1, ..., x_n.”

Inference rule:

\[
\begin{align*}
\Delta' & \vdash Q :: (x : A') \\
\Delta & \vdash P; Q :: (x : A)
\end{align*}
\]

Left and right rules:

\[
\begin{align*}
\Delta', x : B & \vdash Q :: (z : C) \\
\Delta, x : A \diamond B & \vdash P; Q :: (z : C)
\end{align*}
\]

\[
\begin{align*}
\Delta' & \vdash Q :: (x : B) \\
\Delta & \vdash P; Q :: (x : A \diamond B)
\end{align*}
\]
Multiplicative conjunction - channel output
Multiplicative conjunction - channel output

\[ \vdash P :: (x : \_ ) \]

\[ \vdash \text{send } x \ y; P :: (x : A \otimes B) \]

\[ \otimes_R \]
Multiplicative conjunction - channel output

\[ \vdash P :: (x : B) \]

\[ \vdash \text{send } x \; y; \; P :: (x : A \otimes B) \]

\[ \otimes_R \]
Multiplicative conjunction - channel output

\[ \Delta, y : A \vdash \text{send } x \; y; \; P :: (x : A \otimes B) \quad \otimes_R \]

\[ \vdash P :: (x : B) \]

\[ \Delta, y : A \vdash \text{send } x \; y; \; P :: (x : A \otimes B) \quad \otimes_R \]
Multiplicative conjunction - channel output

\[
\begin{align*}
\Delta \vdash P :: (x : B) \\
\Delta, y : A \vdash \text{send } x \ y; P :: (x : A \otimes B) & \quad \otimes_R
\end{align*}
\]
Multiplicative conjunction - channel output

we have lost y!

\[
\Delta \vdash P :: (x : B)
\]

\[
\Delta, y : A \vdash \text{send } x \; y; \; P :: (x : A \otimes B) \quad \otimes_R
\]
Multiplicative conjunction - channel output

\[
\Delta \vdash P :: (x : B) \\
\Delta, y : A \vdash \text{send } x \ y; P :: (x : A \otimes B) \quad \otimes_R
\]

\[
\Delta, x : A \otimes B \vdash y \leftarrow \text{recv } x; Q_y :: (z : C') \quad \otimes_L
\]
Multiplicative conjunction - channel output

\[ \Delta \vdash P :: (x : B) \]
\[ \Delta, y : A \vdash \text{send } x \ y; P :: (x : A \otimes B) \]

\[ \otimes_R \]

\[ \Delta, x : B \vdash Q_y :: (z : C) \]
\[ \Delta, x : A \otimes B \vdash y \leftarrow \text{recv } x; Q_y :: (z : C) \]

\[ \otimes_L \]
Multiplicative conjunction - channel output

\[
\begin{align*}
\Delta & \vdash P :: (x : B) \\
\Delta, y : A & \vdash \text{send } x\ y;\ P :: (x : A \otimes B) \quad \otimes_R
\end{align*}
\]

\[
\begin{align*}
\Delta, x : B, y : A & \vdash Q_y :: (z : C) \\
\Delta, x : A \otimes B & \vdash y \leftarrow \text{recv } x;\ Q_y :: (z : C) \quad \otimes_L
\end{align*}
\]
Multiplicative implication - channel input
Multiplicative implication - channel input

\[
\Delta \vdash P_y :: (x : ) \\
\Delta \vdash y \leftarrow \text{recv } x; P_y :: (x : A \multimap B) \overset{\circ R}{\longrightarrow}
\]
Multiplicative implication - channel input

\[
\begin{align*}
\Delta & \vdash P_y :: (x : B) \\
\Delta & \vdash y \leftarrow \text{recv } x; P_y :: (x : A \rightarrow B) \quad \circ_R
\end{align*}
\]
Multiplicative implication - channel input

\[ \frac{\Delta, y : A \vdash P_y :: (x : B)}{\Delta \vdash y \leftarrow \text{recv } x; P_y :: (x : A \multimap B)} \]
Multiplicative implication - channel input

\[
\begin{align*}
\Delta, y : A &\vdash P_y :: (x : B) \\
\Delta &\vdash y \leftarrow \text{recv } x; P_y :: (x : A \to B) & \circ_R \\
\Delta, x : &\vdash Q :: (z : C) \\
\Delta, x : A \to B &\vdash \text{send } x y; Q :: (z : C) & \circ_L
\end{align*}
\]
Multiplicative implication - channel input

\[
\begin{align*}
\Delta, y : A &\vdash P_y :: (x : B) \\
\Delta &\vdash y \leftarrow \text{recv } x; P_y :: (x : A \rightarrow B) & \quad \circ_R
\end{align*}
\]

\[
\begin{align*}
\Delta, x : B &\vdash Q :: (z : C) \\
\Delta, x : A \rightarrow B &\vdash \text{send } x \ y; Q :: (z : C) & \quad \circ_L
\end{align*}
\]
Multiplicative implication - channel input

\[
\Delta, y : A \vdash P_y :: (x : B) \\
\Delta \vdash y \leftarrow \text{recv } x; P_y :: (x : A \to B) \quad \text{``}_R
\]

\[
\Delta, x : B \vdash Q :: (z : C) \\
\Delta, x : A \to B, y : A \vdash \text{send } x y; Q :: (z : C) \quad \text{``}_L
\]
Multiplicative implication - channel input

\[ \Delta, y : A \vdash P_y :: (x : B) \]
\[ \Delta \vdash y \leftarrow \text{recv } x ; P_y :: (x : A \rightarrow B) \]

we have lost y!

\[ \Delta, x : B \vdash Q :: (z : C) \]
\[ \Delta, x : A \rightarrow B, y : A \vdash \text{send } x \ y ; Q :: (z : C) \]
Additive disjunction — internal choice
Additive disjunction — internal choice

\[
\frac{P :: (x : )}{\Delta \vdash x.\text{inl}; P :: (x : A \oplus B)} \quad \oplus_{R_1}
\]

\[
\frac{P :: (x : )}{\Delta \vdash x.\text{inr}; P :: (x : A \oplus B)} \quad \oplus_{R_2}
\]
Additive disjunction — internal choice

\[ \frac{\vdash P :: (x : A)}{\Delta \vdash x.\text{inl}; P :: (x : A \oplus B)} \oplus_{R_1} \]

\[ \frac{\vdash P :: (x : )}{\Delta \vdash x.\text{inr}; P :: (x : A \oplus B)} \oplus_{R_2} \]
Additive disjunction — internal choice

\[
\begin{align*}
\vdash P :: (x : A) \\
\Delta \vdash x.\text{inl}; P :: (x : A \oplus B) & \quad \oplus R_1 \\
\vdash P :: (x : B) \\
\Delta \vdash x.\text{inr}; P :: (x : A \oplus B) & \quad \oplus R_2
\end{align*}
\]
Additive disjunction — internal choice

\[
\begin{align*}
\Delta & \vdash P :: (x : A) \\
\Delta & \vdash x.\text{inl}; P :: (x : A \oplus B) & \oplus R_1 \\
\Delta & \vdash P :: (x : B) \\
\Delta & \vdash x.\text{inr}; P :: (x : A \oplus B) & \oplus R_2
\end{align*}
\]
Additive disjunction — internal choice

\[
\Delta \vdash P :: (x : A) \\
\Delta \vdash x.\text{inl}; P :: (x : A \oplus B) \quad \oplus_{R_1}
\]

\[
\Delta \vdash P :: (x : B) \\
\Delta \vdash x.\text{inr}; P :: (x : A \oplus B) \quad \oplus_{R_2}
\]

\[
\Delta, x : \quad \vdash Q_1 :: (z : C) \quad \Delta, x : \quad \vdash Q_2 :: (z : C) \\
\Delta, x : A \oplus B \vdash \text{case } x \text{ of } (Q_1, Q_2) :: (z : C) \quad \oplus_{L}
\]
Additive disjunction — internal choice

\[
\begin{align*}
\Delta & \vdash P :: (x : A) \\
\Delta & \vdash x.\text{inl}; P :: (x : A \oplus B) \quad \oplus_{R_1}
\end{align*}
\]

\[
\begin{align*}
\Delta & \vdash P :: (x : B) \\
\Delta & \vdash x.\text{inr}; P :: (x : A \oplus B) \quad \oplus_{R_2}
\end{align*}
\]

\[
\begin{align*}
\Delta, x : A & \vdash Q_1 :: (z : C) \\
\Delta, x : A \oplus B & \vdash \text{case } x \text{ of } (Q_1, Q_2) :: (z : C) \quad \oplus_L
\end{align*}
\]
Additive disjunction — internal choice

\[ \begin{align*} 
\Delta \vdash P :: (x : A) \\
\Delta \vdash x.\text{inl}; P :: (x : A \oplus B) \quad \oplus_{R_1} \\
\Delta \vdash P :: (x : B) \\
\Delta \vdash x.\text{inr}; P :: (x : A \oplus B) \quad \oplus_{R_2} \\
\Delta, x : A \vdash Q_1 :: (z : C) \\
\Delta, x : B \vdash Q_2 :: (z : C) \\
\Delta, x : A \oplus B \vdash \text{case } x \text{ of } (Q_1, Q_2) :: (z : C) \quad \oplus_{L} 
\end{align*} \]
Additive disjunction — internal choice

\[ \Delta \vdash P :: (x : A) \]
\[ \Delta \vdash x.\text{inl}; P :: (x : A \oplus B) \overset{\oplus R_1}{\Rightarrow} \]

\[ \Delta \vdash P :: (x : B) \]
\[ \Delta \vdash x.\text{inr}; P :: (x : A \oplus B) \overset{\oplus R_2}{\Rightarrow} \]

\[ \Delta, x : A \vdash Q_1 :: (z : C) \quad \Delta, x : B \vdash Q_2 :: (z : C) \]
\[ \Delta, x : A \oplus B \vdash \text{case } x \text{ of } (Q_1, Q_2) :: (z : C) \overset{\oplus L}{\Rightarrow} \]

internal choice: provider chooses. Generalize to n-ary choice
Additive conjunction — external choice
Additive conjunction — external choice

\[
\vdash P_1 :: (x : ~) \quad \vdash P_2 :: (x : ~) \\
\Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \& B) \quad \&_R
\]
Additive conjunction — external choice

\[ \vdash P_1 :: (x : A) \quad \vdash P_2 :: (x : B) \]
\[ \Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \& B) \quad \&_R \]
Additive conjunction — external choice

\[
\begin{align*}
\vdash P_1 :: (x : A) & \quad \vdash P_2 :: (x : B) \\
\Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \& B) & \quad \& R
\end{align*}
\]
Additive conjunction — external choice

\[\Delta \vdash P_1 :: (x : A) \quad \Delta \vdash P_2 :: (x : B)\]

\[\Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \& B) \quad \&_R\]
Additive conjunction — external choice

\[ \Delta \vdash P_1 :: (x : A) \quad \Delta \vdash P_2 :: (x : B) \]

\[ \Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \land B) \quad &_R \]

\[ \Delta, x : \quad \vdash Q :: (z : C) \]

\[ \Delta, x : A \land B \vdash x.\text{inl} ; Q :: (z : C) \quad &_{L_1} \]

\[ \Delta, x : \quad \vdash Q :: (z : C) \]

\[ \Delta, x : A \land B \vdash x.\text{inr} ; Q :: (z : C) \quad &_{L_2} \]
Additive conjunction — external choice

\[
\Delta \vdash P_1 :: (x : A) \quad \Delta \vdash P_2 :: (x : B) \\
\Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \& B) \quad \&_R
\]

\[
\Delta, x : A \vdash Q :: (z : C) \\
\Delta, x : A \& B \vdash x.\text{inl}; Q :: (z : C) \quad \&_{L_1}
\]

\[
\Delta, x : \quad \vdash Q :: (z : C) \\
\Delta, x : A \& B \vdash x.\text{inr}; Q :: (z : C) \quad \&_{L_2}
\]
Additive conjunction — external choice

\[
\frac{\Delta \vdash P_1 :: (x : A) \quad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \& B)} \quad &_R
\]

\[
\frac{\Delta, x : A \vdash Q :: (z : C)}{\Delta, x : A \& B \vdash x.\text{inl}; Q :: (z : C)} \quad &_{L_1}
\]

\[
\frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \& B \vdash x.\text{inr}; Q :: (z : C)} \quad &_{L_2}
\]
Additive conjunction — external choice

\[
\begin{align*}
\Delta \vdash P_1 :: (x : A) & \quad \Delta \vdash P_2 :: (x : B) \\
\Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \& B) & \quad \&_R
\end{align*}
\]

\[
\begin{align*}
\Delta, x : A \vdash Q :: (z : C) & \\
\Delta, x : A \& B \vdash x.\text{inl}; Q :: (z : C) & \quad \&_{L_1}
\end{align*}
\]

\[
\begin{align*}
\Delta, x : B \vdash Q :: (z : C) & \\
\Delta, x : A \& B \vdash x.\text{inr}; Q :: (z : C) & \quad \&_{L_2}
\end{align*}
\]

external choice: client chooses. Generalize to n-ary choice