Verificación de Programas Concurrentes Usando Lógica
Logical Foundations of Concurrent Computation

Jorge A. Pérez
University of Groningen, The Netherlands
j.a.perez [[at]] rug.nl

ECI 2026
(Version of July 28, 2026)

Propositions as Sessions

Outline

A Language of Processes

Assume a set of **names**/**channels** $x, y, z, \ldots$, we define **processes**:

A Language of Processes

Assume a set of **names**/**channels** $x, y, z, \ldots$, we define **processes**:

$$
\begin{array}{rll}
P, Q ::= & \overline{x}\langle y \rangle.P & \textbf{send } y \text{ along channel/session } x, \text{ continue as } P \\
\mid & x(y).P & \textbf{receive } z \text{ along } x, \text{ continue as } P\{^z/_y\}
\end{array}
$$

A Language of Processes

Assume a set of **names**/**channels** $x, y, z, \ldots$, we define **processes**:

$$
\begin{array}{rll}
P, Q ::= & \overline{x}\langle y \rangle.P & \textbf{send } y \text{ along channel/session } x, \text{ continue as } P \\
\mid & x(y).P & \textbf{receive } z \text{ along } x, \text{ continue as } P\{^{z}/_{y}\} \\
\mid & \overline{x}\langle\rangle.P & \textbf{close } x, \text{ continue as } P \\
\mid & x().P & \textbf{wait-close } x, \text{ continue as } P
\end{array}
$$

A Language of Processes

Assume a set of **names**/**channels** $x, y, z, \ldots$, we define **processes**:

$P, Q ::=$	$\overline{x}\langle y\rangle.P$	**send** y along channel/session x, continue as P
$\mid$	$x(y).P$	**receive** z along x, continue as $P\{^z/_y\}$
$\mid$	$\overline{x}\langle\rangle.P$	**close** x, continue as P
$\mid$	$x().P$	**wait-close** x, continue as P
$\mid$	$P \mid Q$	**parallel composition of** P and Q
$\mid$	$(\boldsymbol{\nu} x)P$	**restriction**: x is local to P
$\mid$	0	**inaction**

A Language of Processes

Assume a set of **names**/**channels** $x, y, z, \ldots$, we define **processes**:

$P, Q ::=$	$\overline{x}\langle y\rangle.P$	**send** y along channel/session x, continue as P
$\mid$	$x(y).P$	**receive** z along x, continue as $P\{^z/_y\}$
$\mid$	$\overline{x}\langle\rangle.P$	**close** x, continue as P
$\mid$	$x().P$	**wait-close** x, continue as P
$\mid$	$P \mid Q$	**parallel composition of** P and Q
$\mid$	$(\boldsymbol{\nu} x)P$	**restriction**: x is local to P
$\mid$	0	**inaction**

This is a variant of the **π-calculus** (Milner, Parrow & Walker, 1992).

- In $(\boldsymbol{\nu} y)P$ and $x(y).P$, name y is **bound** with scope P.
- We write $P\{^y/_z\}$ to denote the **substitution** of z for y in P.

Process Synchronizations, Informally

- A ‘basic’ reduction step:

$$\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow P \mid Q\{y/z\}$$

Process Synchronizations, Informally

- A 'basic' reduction step:

$$\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow P \mid Q\{y/z\}$$

- Consider the process $(\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q$. Name y is bound (a 'secret' to P).

Process Synchronizations, Informally

- A 'basic' reduction step:

$$\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow P \mid Q\{y/z\}$$

- Consider the process $(\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q$. Name y is bound (a 'secret' to P). We have a synchronization:

$$(\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow (\nu y)(P \mid Q\{y/z\})$$

  Observe: The scope of the local name y **extends** after reduction!

Process Synchronizations, Informally

- A 'basic' reduction step:

$$\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow P \mid Q\{y/z\}$$

- Consider the process $(\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q$. Name y is bound (a 'secret' to P). We have a synchronization:

$$(\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow (\nu y)(P \mid Q\{y/z\})$$

  Observe: The scope of the local name y **extends** after reduction!

- Now consider $(\nu x)((\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q) \mid x(u).R$.
  Both the message y and the channel x are bound.

Process Synchronizations, Informally

- A 'basic' reduction step:

$$\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow P \mid Q\{y/z\}$$

- Consider the process $(\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q$. Name y is bound (a 'secret' to P). We have a synchronization:

$$(\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow (\nu y)(P \mid Q\{y/z\})$$

  Observe: The scope of the local name y **extends** after reduction!

- Now consider $(\nu x)((\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q) \mid x(u).R$.
  Both the message y and the channel x are bound. We have the reduction:

$$(\nu x)((\nu y)\overline{x}\langle y\rangle.P \mid x(z).Q) \mid x(u).R \longrightarrow (\nu x)(\nu y)(P \mid Q\{y/z\}) \mid x(u).R$$

Process Synchronizations, Formally

Reduction, written $P \longrightarrow Q$, is a binary relation defined by the rules:

$$\overline{x}\langle z\rangle.P \mid x(y).Q \longrightarrow P \mid Q\{^z/_y\} \qquad \overline{x}\langle\rangle.P \mid x().Q \longrightarrow P \mid Q$$

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \frac{P \longrightarrow P'}{(\boldsymbol{\nu} x)P \longrightarrow (\boldsymbol{\nu} x)P'} \qquad \frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$$

where $\equiv$ is a

Process Synchronizations, Formally

Reduction, written $P \longrightarrow Q$, is a binary relation defined by the rules:

$$\overline{x}\langle z\rangle.P \mid x(y).Q \longrightarrow P \mid Q\{^z/_y\} \qquad \overline{x}\langle\rangle.P \mid x().Q \longrightarrow P \mid Q$$

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \frac{P \longrightarrow P'}{(\boldsymbol{\nu} x)P \longrightarrow (\boldsymbol{\nu} x)P'} \qquad \frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$$

where $\equiv$ is a **structural congruence** on processes defined as follows:

$$P \mid 0 \equiv P \qquad P =_\alpha Q \Rightarrow P \equiv Q$$

$$P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$(\boldsymbol{\nu} x)0 \equiv 0 \qquad x \notin \mathrm{fn}(P) \Rightarrow P \mid (\boldsymbol{\nu} x)Q \equiv (\boldsymbol{\nu} x)(P \mid Q)$$

$$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P$$

Session Types: A Language of Protocols

$$S ::= \; !U; S \;\mid\; ?U; S$$

send value of type U, continue as S

receive value of type U, continue as S

Session Types: A Language of Protocols

$$
\begin{array}{rll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{branching } S_1, \ldots, S_n \text{ (external choice)} \\
 & & \quad (l_1, \ldots, l_n \text{ are } \textbf{labels}) \\
\mid & \oplus\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{select} \text{ (internal choice) between } S_1, \ldots, S_n
\end{array}
$$

Session Types: A Language of Protocols

$$
\begin{array}{lll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{branching } S_1, \ldots, S_n \text{ (external choice)} \\
 & & \quad (l_1, \ldots, l_n \text{ are } \textbf{labels}) \\
\mid & \oplus\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{select} \text{ (internal choice) between } S_1, \ldots, S_n \\
\mid & \mu t.S \mid t & \textbf{recursion}
\end{array}
$$

Session Types: A Language of Protocols

$$
\begin{array}{rll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{branching } S_1, \ldots, S_n \text{ (external choice)} \\
 & & \quad (l_1, \ldots, l_n \text{ are } \textbf{labels}) \\
\mid & \oplus\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{select} \text{ (internal choice) between } S_1, \ldots, S_n \\
\mid & \mu t.S \mid t & \textbf{recursion} \\
\mid & \mathtt{end} & \textbf{terminated protocol}
\end{array}
$$

Session Types: A Language of Protocols

$$
\begin{array}{rll}
S ::= & !U;S & \textbf{send value} \text{ of type } U, \text{ continue as } S \\
\mid & ?U;S & \textbf{receive value} \text{ of type } U, \text{ continue as } S \\
\mid & \&\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{branching } S_1, \ldots, S_n \text{ (external choice)} \\
 & & \quad (l_1, \ldots, l_n \text{ are } \textbf{labels}) \\
\mid & \oplus\{l_1 : S_1, \cdots, l_n : S_n\} & \textbf{select} \text{ (internal choice) between } S_1, \ldots, S_n \\
\mid & \mu t.S \mid t & \textbf{recursion} \\
\mid & \texttt{end} & \textbf{terminated protocol}
\end{array}
$$

Notice:

- Sequential communication patterns (no built-in concurrency).
- U stands for basic values (e.g. `int`) but also other sessions S.

Example: A Two-Buyer Protocol

Alice and **Bob** cooperate in buying a book from **Seller**:

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks whether Bob can contribute in buying the book.
3. Alice uses the answer from Bob (yes/no) to interact with Seller, either:
   a) completing the payment and arranging delivery details
   b) canceling the transaction
4. In case 3(a) Alice contacts Bob to get his address, and forwards it to Seller.

4'. In case 3(b) Alice is in charge of gracefully concluding the conversation.

Note: The structure of messaging matters!

Session Types for The Two-Buyer Protocol

Two independent protocols, with Alice “leading” the interactions:

Session Types for The Two-Buyer Protocol

Two independent protocols, with Alice "leading" the interactions:

1. A session type for Seller (in its interaction with Alice):

$$S_{\mathsf{SA}} = \texttt{?book};\ \texttt{!quote};\ \& \begin{cases} \text{buy} : & \texttt{?paym};\ \texttt{?address};\ \texttt{!ok};\ \text{end} \\ \text{cancel} : & \texttt{?thanks};\ \texttt{!bye};\ \text{end} \end{cases}$$

2. A session type for Alice (in its interaction with Bob):

$$S_{\mathsf{AB}} = \texttt{!cost};\ \& \begin{cases} \text{share} : & \texttt{?address};\ \texttt{!ok};\ \text{end} \\ \text{close} : & \texttt{!bye};\ \text{end} \end{cases}$$

Example: A Two-Buyer Protocol

Correctness follows from the interplay of the following properties:

- **Fidelity** – implementations **follow the intended protocol**.

Example: A Two-Buyer Protocol

Correctness follows from the interplay of the following properties:

- **Fidelity** – implementations **follow the intended protocol**.
- **Safety** – they don't feature **communication errors**.

Example: A Two-Buyer Protocol

Correctness follows from the interplay of the following properties:

- **Fidelity** – implementations **follow the intended protocol**.
- **Safety** – they don't feature **communication errors**.
- **Deadlock-Freedom** – they do not **"get stuck"** while running the protocol.

Example: A Two-Buyer Protocol

Correctness follows from the interplay of the following properties:

- **Fidelity** – implementations **follow the intended protocol**.
- **Safety** – they don't feature **communication errors**.
- **Deadlock-Freedom** – they do not **"get stuck"** while running the protocol.
- **Termination** – they do not engage in **infinite behavior** (that may prevent them from completing the protocol)

Linear Logic: Sequent Calculus

The sequent

$$A_1, \ldots, A_n \vdash B,$$

is interpreted as $A_1 \otimes \cdots \otimes A_n \multimap B$.

Linear Logic: Sequent Calculus

The sequent

$$A_1, \ldots, A_n \vdash B,$$

is interpreted as $A_1 \otimes \cdots \otimes A_n \multimap B$.

Structural rules:

$$\frac{}{A \vdash A} \qquad \frac{\Delta_1 \vdash A \qquad \Delta_2, A \vdash B}{\Delta_1, \Delta_2 \vdash B} \qquad \frac{\Delta_1, A, B, \Delta_2 \vdash C}{\Delta_1, B, A, \Delta_2 \vdash C}$$

Notice: In sequent calculus, each connective is "explained" with
- a left rule,
- a right rule, and
- the interactions with the cut rule.

MAILL: Multiplicative Additive Intuitionistic LL

$$A, B \;::=\; 1 \mid A \otimes B \mid A \multimap B \mid A \,\&\, B \mid A \oplus B$$

$$\frac{}{A \vdash A} \qquad \frac{}{\emptyset \vdash 1}$$

$$\otimes\mathrm{R}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B} \qquad \otimes\mathrm{L}\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$

$$\mathrm{Cut}\ \frac{\Gamma \vdash A \qquad \Gamma', A \vdash B}{\Gamma, \Gamma' \vdash B} \qquad \multimap\mathrm{R}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \multimap\mathrm{L}\ \frac{\Gamma_1 \vdash A \qquad \Gamma_2, B \vdash C}{\Gamma_1, \Gamma_2, A \multimap B \vdash C}$$

$$\&\mathrm{R}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B} \qquad \&\mathrm{L}_i\ \frac{\Gamma, A_i \vdash C}{\Gamma, A_1 \,\&\, A_2 \vdash C} \qquad \oplus\mathrm{R}_i\ \frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 \oplus A_2} \qquad \oplus\mathrm{L}\ \frac{\Gamma, A_1 \vdash C \qquad \Gamma, A_2 \vdash C}{\Gamma, A_1 \oplus A_2 \vdash C}$$

Outline

Propositions As Types

The 'standard', sequential case (aka Curry-Howard correspondence, formulae-as-types, proofs-as-programs, ...):

Intuitionistic logic propositions	$\Longleftrightarrow$	types describing **data**
Natural deduction derivations	$\Longleftrightarrow$	**terms** in the λ-calculus
Proof reductions	$\Longleftrightarrow$	β-reductions

Propositions As Sessions

Today, the concurrent case:

Linear logic propositions	$\Longleftrightarrow$	types describing **behavior** (sessions)
Sequent calculus derivations	$\Longleftrightarrow$	**processes** in the π-calculus
Cut reductions	$\Longleftrightarrow$	communication between processes

Propositions As Sessions

Today, the concurrent case:

Linear logic propositions	$\Longleftrightarrow$	types describing **behavior** (sessions)
Sequent calculus derivations	$\Longleftrightarrow$	**processes** in the π-calculus
Cut reductions	$\Longleftrightarrow$	communication between processes

- We shall follow the correspondence between session types and intuitionistic LL by Caires & Pfenning (2010).
- Also relevant: the correspondence with classical LL by Wadler (2012).

Key Ideas

- We shall consider a language of **processes** (denoted $P, Q, \ldots$) that interact by synchronizing on **names** (denoted $x, y, z, \ldots$).

Key Ideas

- We shall consider a language of **processes** (denoted $P, Q, \ldots$) that interact by synchronizing on **names** (denoted $x, y, z, \ldots$).
- Processes can send/receive messages on names, using sequencing and parallel composition. We shall gradually "**extract**" their syntax from proofs.

Key Ideas

- Interpret the logical sequent

$$A_1, \cdots, A_n \vdash C$$

as a **typing judgment**, under a suitable reading of propositions as sessions:

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: z : C$$

Key Ideas

- Interpret the logical sequent

$$A_1, \cdots, A_n \vdash C$$

as a **typing judgment**, under a suitable reading of propositions as sessions:

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: z : C$$

Process P **offers** protocol C on channel z...

Key Ideas

- Interpret the logical sequent

$$A_1, \cdots, A_n \vdash C$$

as a **typing judgment**, under a suitable reading of propositions as sessions:

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: z : C$$

Process P **offers** protocol C on channel z...

... by **relying on** protocols $A_1, \ldots, A_n$ on channels $x_1, \ldots, x_n$.

The **Typed** Process Language, courtesy of Curry-Howard

$$P, Q ::= \quad (\boldsymbol{\nu} y)\,(\overline{x}\langle y\rangle.(P \mid Q)) \qquad \textbf{send } y \text{ over } x\text{, then execute } P \text{ and } Q$$

$$\mid \quad x(y).P \qquad \textbf{receive } y \text{ over } x\text{, then execute } P$$

The **Typed** Process Language, courtesy of Curry-Howard

$P, Q ::=$	$(\boldsymbol{\nu} y)\,(\overline{x}\langle y\rangle.(P \mid Q))$	**send** y over x, then execute P and Q
$\mid$	$x(y).P$	**receive** y over x, then execute P
$\mid$	$\overline{x}\langle\rangle$	**close** session x
$\mid$	$x().P$	**wait-close** session x, then execute P

The **Typed** Process Language, courtesy of Curry-Howard

$P, Q ::=$	$(\boldsymbol{\nu} y)\,(\overline{x}\langle y\rangle.(P \mid Q))$	**send** y over x, then execute P and Q
$\mid$	$x(y).P$	**receive** y over x, then execute P
$\mid$	$\overline{x}\langle\rangle$	**close** session x
$\mid$	$x().P$	**wait-close** session x, then execute P
$\mid$	$x \triangleleft \mathtt{l}_i; P$	**select** label $\mathtt{l}_i$ along x, then execute P
$\mid$	$x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\}$	**branch** on x, offering labels $\mathtt{l}_1, \ldots, \mathtt{l}_n$

The **Typed** Process Language, courtesy of Curry-Howard

$P, Q ::=$	$(\boldsymbol{\nu} y)\,(\overline{x}\langle y\rangle.(P \mid Q))$	**send** y over x, then execute P and Q
$\mid$	$x(y).P$	**receive** y over x, then execute P
$\mid$	$\overline{x}\langle\rangle$	**close** session x
$\mid$	$x().P$	**wait-close** session x, then execute P
$\mid$	$x \triangleleft \mathtt{l}_i; P$	**select** label $\mathtt{l}_i$ along x, then execute P
$\mid$	$x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\}$	**branch** on x, offering labels $\mathtt{l}_1, \ldots, \mathtt{l}_n$
$\mid$	$(\boldsymbol{\nu} x)(P \mid Q)$	**parallel composition** of P and Q on x

The **Typed** Process Language, courtesy of Curry-Howard

$P, Q ::=$	$(\boldsymbol{\nu} y)\,(\overline{x}\langle y\rangle.(P \mid Q))$	**send** y over x, then execute P and Q
$\mid$	$x(y).P$	**receive** y over x, then execute P
$\mid$	$\overline{x}\langle\rangle$	**close** session x
$\mid$	$x().P$	**wait-close** session x, then execute P
$\mid$	$x \triangleleft \mathtt{l}_i; P$	**select** label $\mathtt{l}_i$ along x, then execute P
$\mid$	$x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\}$	**branch** on x, offering labels $\mathtt{l}_1, \ldots, \mathtt{l}_n$
$\mid$	$(\boldsymbol{\nu} x)(P \mid Q)$	**parallel composition** of P and Q on x
$\mid$	$[y \leftarrow x]$	**forwarder** between sessions x and y

The **Typed** Process Language, courtesy of Curry-Howard

$P, Q ::=$	$(\boldsymbol{\nu} y)\,(\overline{x}\langle y\rangle.(P \mid Q))$	**send** y over x, then execute P and Q
$\mid$	$x(y).P$	**receive** y over x, then execute P
$\mid$	$\overline{x}\langle\rangle$	**close** session x
$\mid$	$x().P$	**wait-close** session x, then execute P
$\mid$	$x \triangleleft \mathtt{l}_i; P$	**select** label $\mathtt{l}_i$ along x, then execute P
$\mid$	$x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\}$	**branch** on x, offering labels $\mathtt{l}_1, \ldots, \mathtt{l}_n$
$\mid$	$(\boldsymbol{\nu} x)(P \mid Q)$	**parallel composition** of P and Q on x
$\mid$	$[y \leftarrow x]$	**forwarder** between sessions x and y
$\mid$	0	**inaction** (silent interpretation)

The **Typed** Process Language, courtesy of Curry-Howard

- As we will see, the process language is **typed** in the sense that the type system (sequent calculus) will exclude certain process forms.

The **Typed** Process Language, courtesy of Curry-Howard

- As we will see, the process language is **typed** in the sense that the type system (sequent calculus) will exclude certain process forms.
- For instance, the process $P \mid Q$ will not be typable.
  Instead, typing only allows $(\boldsymbol{\nu x})(P \mid Q)$, with conditions on P and Q.

The **Typed** Process Language, courtesy of Curry-Howard

- As we will see, the process language is **typed** in the sense that the type system (sequent calculus) will exclude certain process forms.
- For instance, the process $P \mid Q$ will not be typable.
  Instead, typing only allows $(\boldsymbol{\nu x})(P \mid Q)$, with conditions on P and Q.
- Similarly, the process $\overline{x}\langle y \rangle.P$ will not be typable.
  Instead, the process $(\boldsymbol{\nu y})\overline{x}\langle y \rangle.(P \mid Q)$ is allowed, with conditions on P and Q.

Interpreting Propositions as Session Types

$!U;S$	**send value** of type U, continue as S	??
$?U;S$	**receive value** of type U, continue as S	??
$\mathtt{end}$	**terminate** the session	??

Interpreting Propositions as Session Types

$!U;S$	**send value** of type U, continue as S	$U \otimes S$
$?U;S$	**receive value** of type U, continue as S	??
`end`	**terminate** the session	??

Notice: A non-commutative reading of $\otimes$!

Interpreting Propositions as Session Types

$!U;S$	**send value** of type U, continue as S	$U \otimes S$
$?U;S$	**receive value** of type U, continue as S	$U \multimap S$
`end`	**terminate** the session	??

Notice: A non-commutative reading of $\otimes$!

Interpreting Propositions as Session Types

$!U;S$	**send value** of type U, continue as S	$U \otimes S$
$?U;S$	**receive value** of type U, continue as S	$U \multimap S$
`end`	**terminate** the session	1

Notice: A non-commutative reading of $\otimes$!

Examples

Let's consider again our typing judgment by looking at some examples:

$$x_1 : A, x_2 : B \vdash P :: y : C \otimes D$$

$$\emptyset \vdash Q :: y : A \multimap B$$

$$x : A \otimes B, y : C \otimes D \vdash R :: z : 1$$

Examples

Let's consider again our typing judgment by looking at some examples:

$x_1 : A, x_2 : B \vdash P :: y : C \otimes D$

Process P offers an output protocol along y, **provided that** its environment provides session protocols A and B along x_1 and x_2, respectively.

$\emptyset \vdash Q :: y : A \multimap B$

$x : A \otimes B, y : C \otimes D \vdash R :: z : 1$

Examples

Let's consider again our typing judgment by looking at some examples:

$x_1 : A, x_2 : B \vdash P :: y : C \otimes D$

$\emptyset \vdash Q :: y : A \multimap B$

Process Q offers an input protocol along y, without requiring anything from its environment.

$x : A \otimes B, y : C \otimes D \vdash R :: z : 1$

Examples

Let's consider again our typing judgment by looking at some examples:

$x_1 : A, x_2 : B \vdash P :: y : C \otimes D$

$\emptyset \vdash Q :: y : A \multimap B$

$x : A \otimes B, y : C \otimes D \vdash R :: z : 1$

Process R does not offer visible behavior z and yet depends on two different protocols from its environment, on names x and y.

Propositions as Session Types: $A \otimes B$

We have some decisions to make:

$$\frac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1, \Delta_2 \vdash ?? :: ?? : A \otimes B}$$

$$\frac{\Delta, y : A, x : B \vdash R :: z : C}{\Delta, ?? : A \otimes B \vdash ?? :: z : C}$$

Propositions as Session Types: $A \otimes B$

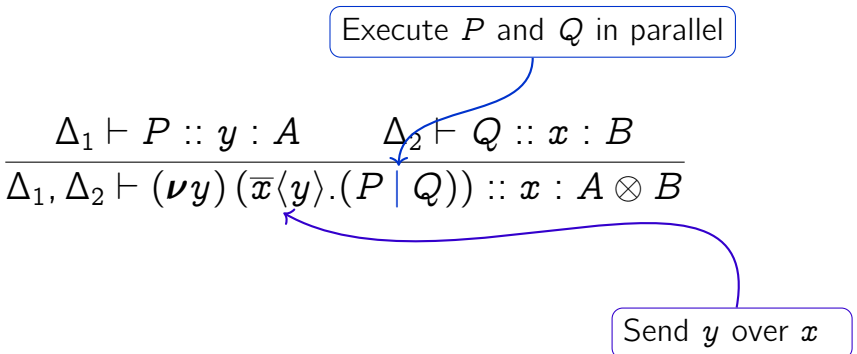
$$\frac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} y)\,(\overline{x}\langle y \rangle.(P \mid Q)) :: x : A \otimes B}$$

Propositions as Session Types: $A \otimes B$

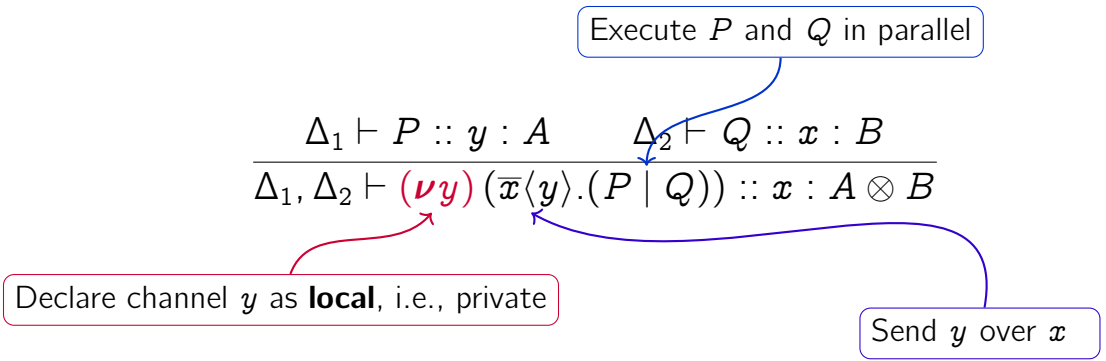
$$\frac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} y)\,(\overline{x}\langle y \rangle.(P \mid Q)) :: x : A \otimes B}$$

Send y over x

Propositions as Session Types: $A \otimes B$



Propositions as Session Types: $A \otimes B$



Propositions as Session Types: $A \otimes B$

$$\frac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} y)\,(\overline{x}\langle y \rangle.(P \mid Q)) :: x : A \otimes B}$$

$$\frac{\Delta, y : A, x : B \vdash R :: z : C}{\Delta, x : A \otimes B \vdash x(y).R :: z : C}$$

To **use** a '$\otimes$', receive a name on x

Propositions as Session Types: $A \multimap B$

For $A \multimap B$, we have a symmetric situation:

$$\frac{\Delta, y : A \vdash P :: z : B}{\Delta \vdash z(y).P :: z : A \multimap B}$$

$$\frac{\Delta_1 \vdash P :: y : A \qquad x : B, \Delta_2 \vdash Q :: z : C}{\Delta_1, x : A \multimap B, \Delta_2 \vdash (\boldsymbol{\nu} y)\,(\overline{x}\langle y \rangle.(P \mid Q)) :: z : C}$$

Propositions as Session Types: $A \multimap B$

For $A \multimap B$, we have a symmetric situation:

$$\frac{\Delta, y : A \vdash P :: z : B}{\Delta \vdash z(y).P :: z : A \multimap B}$$

$$\frac{\Delta_1 \vdash P :: y : A \qquad x : B, \Delta_2 \vdash Q :: z : C}{\Delta_1, x : A \multimap B, \Delta_2 \vdash (\boldsymbol{\nu} y)\,(\overline{x}\langle y \rangle.(P \mid Q)) :: z : C}$$

To **offer** a '$\multimap$', we implement a receive

Propositions as Session Types: $A \multimap B$

For $A \multimap B$, we have a symmetric situation:

$$\frac{\Delta, y : A \vdash P :: z : B}{\Delta \vdash z(y).P :: z : A \multimap B}$$

$$\frac{\Delta_1 \vdash P :: y : A \qquad x : B, \Delta_2 \vdash Q :: z : C}{\Delta_1, x : A \multimap B, \Delta_2 \vdash (\boldsymbol{\nu} y)\,(\overline{x}\langle y \rangle.(P \mid Q)) :: z : C}$$

To **use** a '$\multimap$', we implement a send

To **offer** a '$\multimap$', we implement a receive

Propositions as Session Types: The Unit 1 and Axiom

More decisions to make:

$$\frac{}{\emptyset \vdash ?? :: x : 1} \qquad \frac{\Delta \vdash Q :: z : C}{\Delta, ?? : 1 \vdash ?? :: z : C} \qquad \frac{}{x : A \vdash ?? :: y : A}$$

Propositions as Session Types: The Unit 1 and Axiom

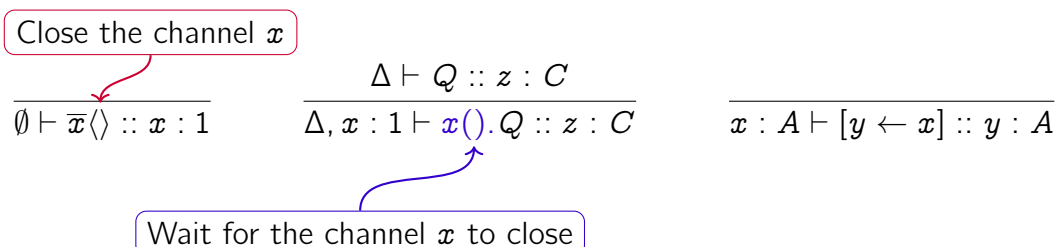
$$\frac{}{\emptyset \vdash \overline{x}\langle\rangle :: x : 1} \qquad \frac{\Delta \vdash Q :: z : C}{\Delta, x : 1 \vdash x().Q :: z : C} \qquad \frac{}{x : A \vdash [y \leftarrow x] :: y : A}$$

Propositions as Session Types: The Unit 1 and Axiom

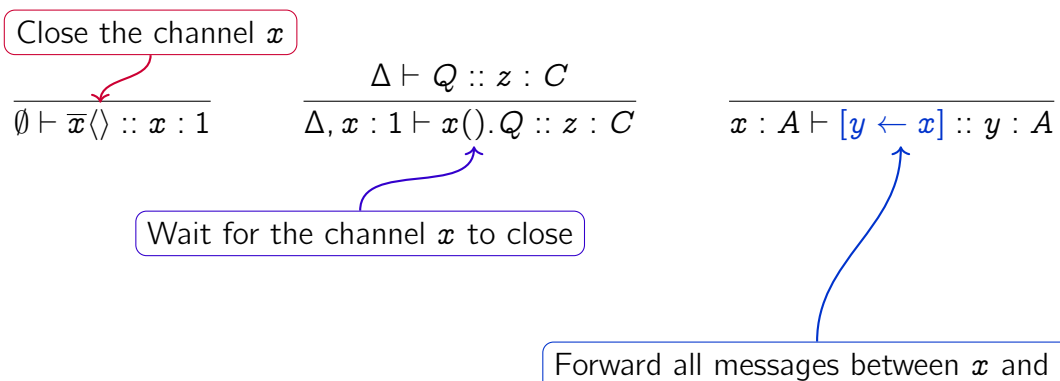
Close the channel x

$$\frac{}{\emptyset \vdash \overline{x}\langle\rangle :: x : 1} \qquad \frac{\Delta \vdash Q :: z : C}{\Delta, x : 1 \vdash x().Q :: z : C} \qquad \frac{}{x : A \vdash [y \leftarrow x] :: y : A}$$

Propositions as Session Types: The Unit 1 and Axiom

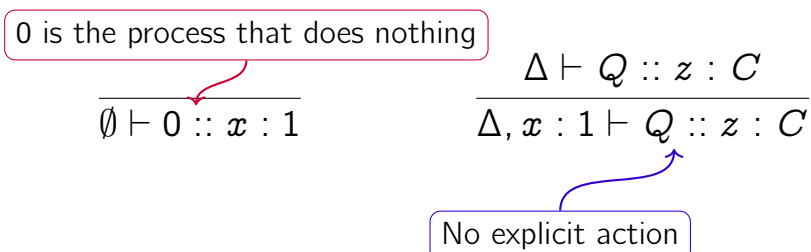


Propositions as Session Types: The Unit 1 and Axiom



Propositions as Session Types: An Alternative for Unit 1

We have just seen an explicit interpretation of 1.
There is also the so-called **silent** interpretation:



Interpreting LL Propositions as Session Types

`end`	**terminate** the session	1
$!U;S$	**send value** of type U, continue as S	$U \otimes S$
$?U;S$	**receive value** of type U, continue as S	$U \multimap S$

Interpreting LL Propositions as Session Types

`end`	**terminate** the session	1
$!U;S$	**send value** of type U, continue as S	$U \otimes S$
$?U;S$	**receive value** of type U, continue as S	$U \multimap S$
$S_1 \oplus S_2$	**select** one between S_1 (left) and S_2 (right)	idem
$S_1 \,\&\, S_2$	**offer** the alternatives S_1 (left) and S_2 (right)	idem

Interpreting LL Propositions as Session Types

$\texttt{end}$	**terminate** the session	1
$!U;S$	**send value** of type U, continue as S	$U \otimes S$
$?U;S$	**receive value** of type U, continue as S	$U \multimap S$
$S_1 \oplus S_2$	**select** one between S_1 (left) and S_2 (right)	idem
$S_1 \,\&\, S_2$	**offer** the alternatives S_1 (left) and S_2 (right)	idem
$\oplus\{l_1 : S_1, \cdots, l_n : S_n\}$	**select** one between $S_1, \ldots, S_n$	"idem"
$\&\{l_1 : S_1, \cdots, l_n : S_n\}$	**offer** the alternatives $S_1, \ldots, S_n$	"idem"

Propositions as Session Types: Additive Conjunction

Binary operators:

$$\frac{\Delta \vdash P :: x : A \qquad \Delta \vdash Q :: x : B}{\Delta \vdash x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} :: x : A \,\&\, B}$$

$$\frac{\Delta, x : A \vdash Q :: z : C}{\Delta, x : A \,\&\, B \vdash x \triangleleft \mathtt{inl}; Q :: z : C} \qquad \frac{\Delta, x : B \vdash Q :: z : C}{\Delta, x : A \,\&\, B \vdash x \triangleleft \mathtt{inr}; Q :: z : C}$$

Notice: '$\triangleleft$' means sending a label and '$\triangleright$' means receiving a label.

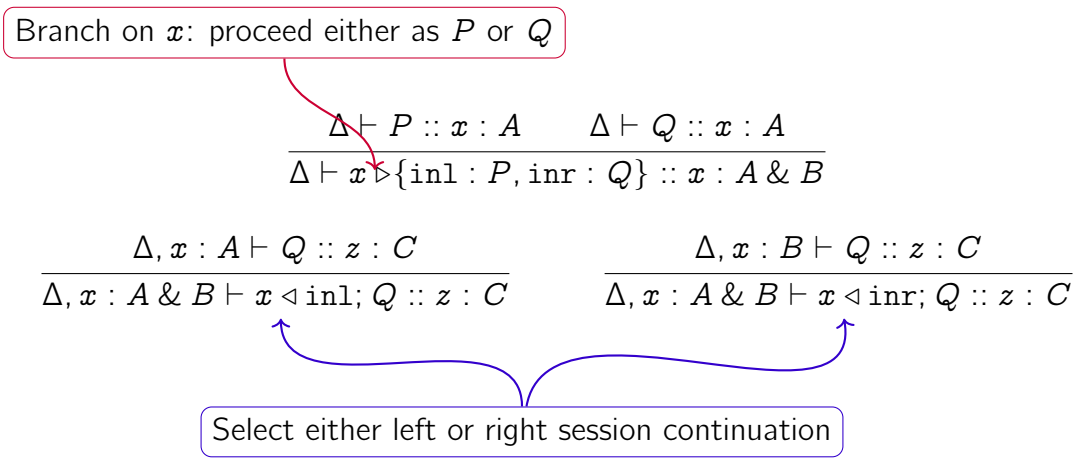
Propositions as Session Types: Additive Conjunction

Branch on x: proceed either as P or Q

$$\frac{\Delta \vdash P :: x : A \qquad \Delta \vdash Q :: x : A}{\Delta \vdash x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} :: x : A \,\&\, B}$$

$$\frac{\Delta, x : A \vdash Q :: z : C}{\Delta, x : A \,\&\, B \vdash x \triangleleft \mathtt{inl}; Q :: z : C} \qquad \frac{\Delta, x : B \vdash Q :: z : C}{\Delta, x : A \,\&\, B \vdash x \triangleleft \mathtt{inr}; Q :: z : C}$$

Propositions as Session Types: Additive Conjunction



Propositions as Session Types: Additive Conjunction

Let's look now at the generalized version:

Branch on x: proceed as one of the P_i

$$\frac{\Delta \vdash P_i :: x : A_i}{\Delta \vdash x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\} :: x : \&\{\mathtt{l}_i : A_i\}_{1 \leq i \leq n}}$$

$$\frac{\Delta, x : A_i \vdash Q :: z : C}{\Delta, x : \&\{\mathtt{l}_i : A_i\} \vdash x \triangleleft \mathtt{l}_i; Q :: z : C}$$

Propositions as Session Types: Additive Conjunction

Let's look now at the generalized version:

Branch on x: proceed as one of the P_i

$$\frac{\Delta \vdash P_i :: x : A_i}{\Delta \vdash x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\} :: x : \&\{\mathtt{l}_i : A_i\}_{1 \le i \le n}}$$

$$\frac{\Delta, x : A_i \vdash Q :: z : C}{\Delta, x : \&\{\mathtt{l}_i : A_i\} \vdash x \triangleleft \mathtt{l}_i ; Q :: z : C}$$

Select exactly one of the session continuations

Propositions as Session Types: Additive Disjunction

For $A \oplus B$, we have a symmetric situation:

$$\frac{\Delta \vdash P :: x : A}{\Delta \vdash x \triangleleft \mathtt{inl}; P :: x : A \oplus B} \qquad \frac{\Delta \vdash Q :: x : B}{\Delta \vdash x \triangleleft \mathtt{inr}; Q :: x : A \oplus B}$$

$$\frac{\Delta, x : A \vdash P :: z : C \qquad \Delta, x : B \vdash Q :: z : C}{\Delta, x : A \oplus B \vdash x \triangleright \{\mathtt{inl} : P; \mathtt{inr} : Q\} :: z : C}$$

Propositions as Session Types: Additive Disjunction

For $A \oplus B$, we have a symmetric situation:

$$\frac{\Delta \vdash P :: x : A}{\Delta \vdash x \triangleleft \mathtt{inl}; P :: x : A \oplus B} \qquad \frac{\Delta \vdash Q :: x : B}{\Delta \vdash x \triangleleft \mathtt{inr}; Q :: x : A \oplus B}$$

$$\frac{\Delta, x : A \vdash P :: z : C \qquad \Delta, x : B \vdash Q :: z : C}{\Delta, x : A \oplus B \vdash x \triangleright \{\mathtt{inl} : P; \mathtt{inr} : Q\} :: z : C}$$

To **offer** a '$\oplus$', we implement a selection

Propositions as Session Types: Additive Disjunction

For $A \oplus B$, we have a symmetric situation:

$$\frac{\Delta \vdash P :: x : A}{\Delta \vdash x \triangleleft \mathtt{inl}; P :: x : A \oplus B} \qquad \frac{\Delta \vdash Q :: x : B}{\Delta \vdash x \triangleleft \mathtt{inr}; Q :: x : A \oplus B}$$

$$\frac{\Delta, x : A \vdash P :: z : C \qquad \Delta, x : B \vdash Q :: z : C}{\Delta, x : A \oplus B \vdash x \triangleright \{\mathtt{inl} : P; \mathtt{inr} : Q\} :: z : C}$$

To **use** a '$\oplus$', we implement a branching

The Process Language, Up to Here

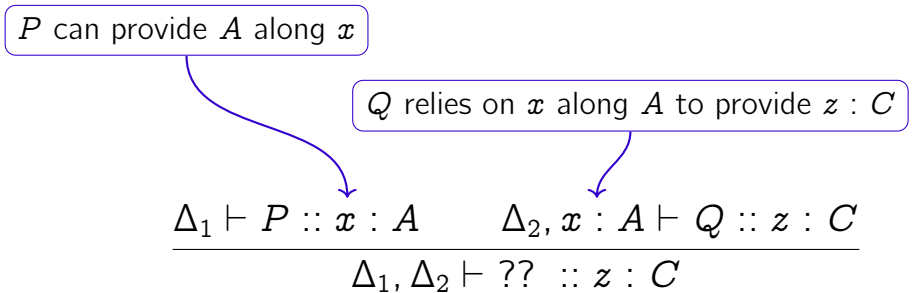
$P, Q ::=$	$[y \leftarrow x]$	**forwarder** between sessions x and y
$\mid$	$(\boldsymbol{\nu} y)\,(\overline{x}\langle y\rangle.(P \mid Q))$	**send** y over x, then execute P and Q
$\mid$	$x(y).P$	**receive** y over x, then execute P
$\mid$	$\overline{x}\langle\rangle$	**close** session x
$\mid$	$x().P$	**wait-close** session x, then execute P
$\mid$	$x \triangleleft \mathtt{l}_i; P$	**select** label $\mathtt{l}_i$ along x, then execute P
$\mid$	$x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\}$	**branch** on x, offering labels $\mathtt{l}_1, \ldots, \mathtt{l}_n$
$\mid$	0	**inaction** (for the silent interpretation)

What are we missing?

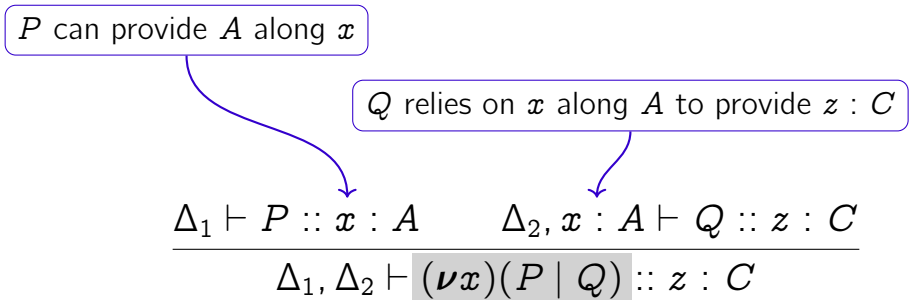
Propositions as Session Types: Cut

$$\frac{\Delta_1 \vdash P :: x : A \qquad \Delta_2, x : A \vdash Q :: z : C}{\Delta_1, \Delta_2 \vdash ?? \ :: z : C}$$

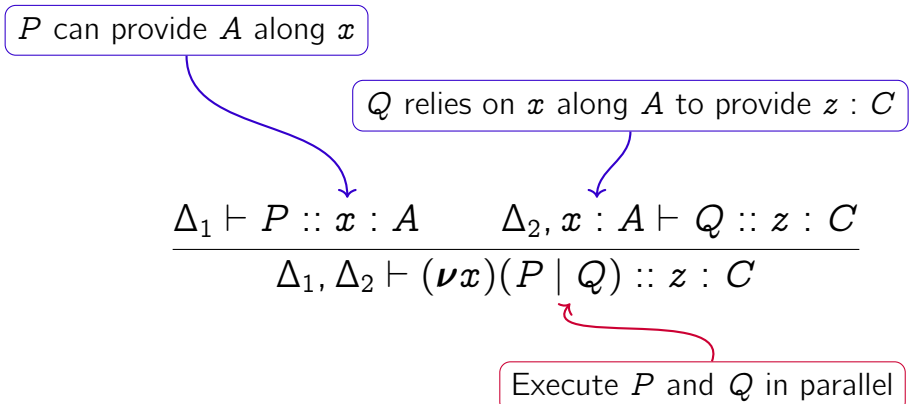
Propositions as Session Types: Cut



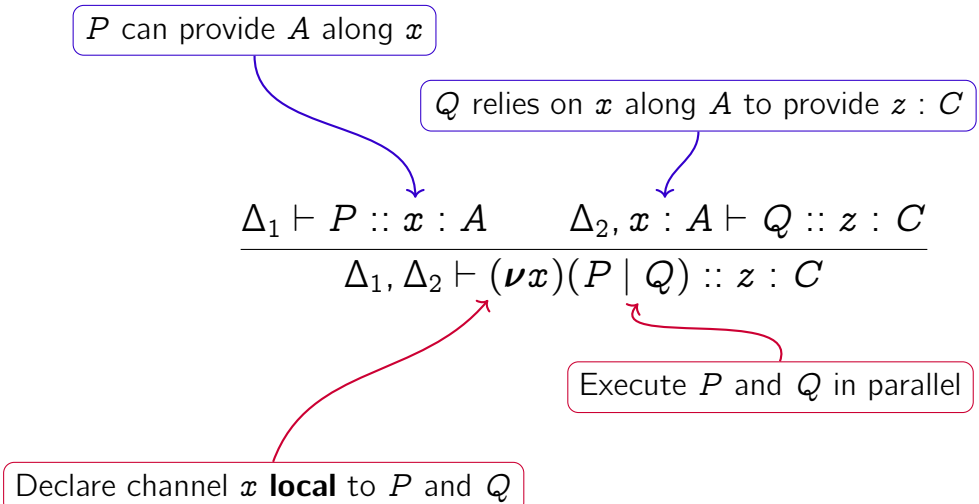
Propositions as Session Types: Cut



Propositions as Session Types: Cut



Propositions as Session Types: Cut



The Process Language, Now With Concurrency

$P, Q ::=$	$[y \leftarrow x]$	**forwarder** between sessions x and y
$\mid$	$(\boldsymbol{\nu} y)\,(\overline{x}\langle y\rangle.(P \mid Q))$	**send** y over x, then execute P and Q
$\mid$	$x(y).P$	**receive** y over x, then execute P
$\mid$	$\overline{x}\langle\rangle$	**close** session x
$\mid$	$x().P$	**wait-close** session x, then execute P
$\mid$	$x \triangleleft \mathtt{l}_i; P$	**select** label $\mathtt{l}_i$ along x, then execute P
$\mid$	$x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\}$	**branch** on x, offering labels $\mathtt{l}_1, \ldots, \mathtt{l}_n$
$\mid$	0	**inaction** (silent interpretation)
$\mid$	$(\boldsymbol{\nu} x)(P \mid Q)$	**parallel composition** of P and Q on x

Open and Closed Systems

- Generally speaking, if we have a process P with judgment

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: y : A$$

P is an **open system**: it can be composed with other processes.
Typing declares $x_1, \ldots, x_n$ as "interfaces".

Open and Closed Systems

- Generally speaking, if we have a process P with judgment

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: y : A$$

  P is an **open system**: it can be composed with other processes.
  Typing declares $x_1, \ldots, x_n$ as "interfaces".
- A process such as $\emptyset \vdash Q :: y : A$ is a **closed system**: its only "interface" is y.

Open and Closed Systems

- Generally speaking, if we have a process P with judgment

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: y : A$$

  P is an **open system**: it can be composed with other processes.
  Typing declares $x_1, \ldots, x_n$ as "interfaces".
- A process such as $\emptyset \vdash Q :: y : A$ is a **closed system**: its only "interface" is y.
- Interpreting cut as typed process composition gives us an immediate recipe for obtaining a closed system. Example:

Open and Closed Systems

- Generally speaking, if we have a process P with judgment

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: y : A$$

  P is an **open system**: it can be composed with other processes.
  Typing declares $x_1, \ldots, x_n$ as "interfaces".
- A process such as $\emptyset \vdash Q :: y : A$ is a **closed system**: its only "interface" is y.
- Interpreting cut as typed process composition gives us an immediate recipe for obtaining a closed system. Example:

$$x_1 : A_1, x_2 : A_2, \cdots, x_n : A_n \vdash P :: y : A$$

Open and Closed Systems

- Generally speaking, if we have a process P with judgment

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: y : A$$

  P is an **open system**: it can be composed with other processes.
  Typing declares $x_1, \ldots, x_n$ as "interfaces".
- A process such as $\emptyset \vdash Q :: y : A$ is a **closed system**: its only "interface" is y.
- Interpreting cut as typed process composition gives us an immediate recipe for obtaining a closed system. Example:

$$x_2 : A_2, \cdots, x_n : A_n \vdash (\boldsymbol{\nu} x_1)(P \mid R_1) :: y : A$$

Open and Closed Systems

- Generally speaking, if we have a process P with judgment

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: y : A$$

  P is an **open system**: it can be composed with other processes.
  Typing declares $x_1, \ldots, x_n$ as "interfaces".
- A process such as $\emptyset \vdash Q :: y : A$ is a **closed system**: its only "interface" is y.
- Interpreting cut as typed process composition gives us an immediate recipe for obtaining a closed system. Example:

$$x_3 : A_3, \cdots, x_n : A_n \vdash (\boldsymbol{\nu} x_2)((\boldsymbol{\nu} x_1)(P \mid R_1) \mid R_2) :: y : A$$

Open and Closed Systems

- Generally speaking, if we have a process P with judgment

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: y : A$$

  P is an **open system**: it can be composed with other processes.
  Typing declares $x_1, \ldots, x_n$ as "interfaces".
- A process such as $\emptyset \vdash Q :: y : A$ is a **closed system**: its only "interface" is y.
- Interpreting cut as typed process composition gives us an immediate recipe for obtaining a closed system. Example:

$$\emptyset \vdash (\boldsymbol{\nu} x_n)(\cdots(\boldsymbol{\nu} x_2)((\boldsymbol{\nu} x_1)(P \mid R_1) \mid R_2) \cdots \mid R_n) :: y : A$$

  where processes R_i offer A_i on x_i, i.e., $\emptyset \vdash R_i :: x_i : A_i$.

Open and Closed Systems

- Generally speaking, if we have a process P with judgment

$$x_1 : A_1, \cdots, x_n : A_n \vdash P :: y : A$$

  P is an **open system**: it can be composed with other processes.
  Typing declares $x_1, \ldots, x_n$ as "interfaces".
- A process such as $\emptyset \vdash Q :: y : A$ is a **closed system**: its only "interface" is y.
- Interpreting cut as typed process composition gives us an immediate recipe for obtaining a closed system. Example:

$$\emptyset \vdash (\boldsymbol{\nu} x_n)(\cdots(\boldsymbol{\nu} x_2)((\boldsymbol{\nu} x_1)(P \mid R_1) \mid R_2) \cdots \mid R_n) :: y : A$$

  where processes R_i offer A_i on x_i, i.e., $\emptyset \vdash R_i :: x_i : A_i$.
- The distinction between open and closed is key in the theory of processes in general, and in the meta-theory of the interpretation in particular.

Example: Processes for The Two-Buyer Protocol

- Recall Bob's involvement in the two-buyer protocol:

$$\texttt{?cost}; \oplus \begin{cases} \text{share}: & \texttt{!address}; \texttt{?ok}; \text{end} \\ \text{close}: & \texttt{?bye}; \text{end} \end{cases}$$

- Here's a possible process implementation for Bob in our process language:

$$\text{Bob} = b(y).b \triangleleft \texttt{share}; (\nu a)\overline{b}\langle a \rangle.([a \leftarrow a'] \mid b(u).0)$$

(This is the silent interpretation!)

Example: Processes for The Two-Buyer Protocol

- Recall Bob's involvement in the two-buyer protocol:

$$?\texttt{cost}; \oplus \begin{cases} \text{share}: & !\texttt{address}; ?\texttt{ok}; \text{end} \\ \text{close}: & ?\texttt{bye}; \text{end} \end{cases}$$

- Here's a possible process implementation for Bob in our process language:

$$\text{Bob} = b(y).b \triangleleft \texttt{share}; (\nu a)\overline{b}\langle a \rangle.([a \leftarrow a'] \mid b(u).0)$$

(This is the silent interpretation!)

- Process Bob is **well-typed**, as the following judgment is provable:

$$a' : \text{A} \vdash \text{Bob} :: b : \underbrace{\text{C} \multimap \oplus\{\texttt{share} : \text{A} \otimes (\text{OK} \multimap 1)\ ;\ \texttt{close} : 1\}}_{\text{bobProto}}$$

where, for simplicity, $\text{C} = \text{A} = \text{OK} = 1$.

Example: Processes for The Two-Buyer Protocol

- Let us now consider an implementation for Alice. She is involved in two different sessions, on which she depends, so we expect a judgment:

$$b : \mathbf{bobProto}, s : \mathbf{sellerProto} \vdash \text{Alice} :: z : 1$$

- We can define the process Alice, which manages both sessions:

$$\overline{s}\langle book\rangle.s(q).\overline{b}\langle q\rangle.b \triangleright \{ \begin{array}{ll} \texttt{share}: & b(addr).s \triangleleft \texttt{buy}; \overline{s}\langle p\rangle.\overline{s}\langle addr\rangle.\overline{b}\langle ok\rangle.0 \\ \texttt{close}: & s \triangleleft \texttt{cancel}; \overline{b}\langle bye\rangle.\overline{s}\langle exit\rangle.0 \end{array} \}$$

- This way, the complete (closed) system would look as follows:

$$(\nu b)((\nu s)(\text{Alice} \mid \text{Seller}) \mid \text{Bob})$$

Outline

Proof Simplification and Process Semantics

- Up to here, we have seen how to interpret propositions as session types, and how to extract processes from proofs.
- This is only half of the expected correspondence.
  We must clarify how proof simplification corresponds to the semantics of processes.

Proof Simplification and Process Semantics

Proof transformations have different consequences on processes:

- Principal cut reductions induce **process reduction**, denoted $\longrightarrow$, a relation that defines the behavior of a process on its own (i.e. synchronizations).

Proof Simplification and Process Semantics

Proof transformations have different consequences on processes:

- Principal cut reductions induce **process reduction**, denoted $\longrightarrow$, a relation that defines the behavior of a process on its own (i.e. synchronizations).
- Some proof transformations correspond to **structural congruence**, denoted $\equiv$, a relation that describes syntactic rearrangements for processes.

Proof Simplification and Process Semantics

Proof transformations have different consequences on processes:

- Principal cut reductions induce **process reduction**, denoted $\longrightarrow$, a relation that defines the behavior of a process on its own (i.e. synchronizations).
- Some proof transformations correspond to **structural congruence**, denoted $\equiv$, a relation that describes syntactic rearrangements for processes.
- Commuting conversions do not have precise correspondences, but can be explained via **behavioral equivalences**.

Principal Cut Reductions: Synchronization via 1 (1/4)

$$\dfrac{\dfrac{}{\emptyset \vdash \overline{x}\langle\rangle :: x : 1} \qquad \dfrac{\Delta \vdash Q :: z : C}{\Delta, x : 1 \vdash x().Q :: z : C}}{\Delta \vdash (\boldsymbol{\nu} x)\Big(\overline{x}\langle\rangle \mid x().Q\Big) :: z : C}$$

$$\longrightarrow$$

Principal Cut Reductions: Synchronization via 1 (2/4)

$$\dfrac{\dfrac{}{\emptyset \vdash \overline{x}\langle\rangle :: x : 1} \qquad \dfrac{\Delta \vdash Q :: z : C}{\Delta, x : 1 \vdash x().Q :: z : C}}{\Delta \vdash (\boldsymbol{\nu} x)\Big(\overline{x}\langle\rangle \mid x().Q\Big) :: z : C}$$

$$\longrightarrow$$

Principal Cut Reductions: Synchronization via 1 (3/4)

$$\dfrac{\dfrac{}{\emptyset \vdash \overline{x}\langle\rangle :: x : 1} \qquad \dfrac{\Delta \vdash Q :: z : C}{\Delta, x : 1 \vdash x().Q :: z : C}}{\Delta \vdash (\nu x)\left(\overline{x}\langle\rangle \mid x().Q\right) :: z : C}$$

$$\longrightarrow$$

$$\Delta \vdash Q :: z : C$$

Principal Cut Reductions: Synchronization via 1 (4/4)

$$\dfrac{\dfrac{}{\emptyset \vdash \overline{x}\langle\rangle :: x : 1} \qquad \dfrac{\Delta \vdash Q :: z : C}{\Delta, x : 1 \vdash x().Q :: z : C}}{\Delta \vdash (\nu x)\Big(\overline{x}\langle\rangle \mid x().Q\Big) :: z : C}$$

$$\longrightarrow$$

$$\Delta \vdash Q :: z : C$$

This way, we have **extracted** the following reduction on processes:

$$(\nu x)\Big(\overline{x}\langle\rangle \mid x().Q\Big) \longrightarrow Q$$

Principal Cut Reductions: Synchronization via $\otimes$

$$\dfrac{\dfrac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) :: x : A \otimes B} \qquad \dfrac{\Delta_3, y : A, x : B \vdash R :: z : C}{\Delta_3, x : A \otimes B \vdash x(y).R :: z : C}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big((\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) \mid x(y).R\Big) :: z : C}$$

$$\longrightarrow$$

Principal Cut Reductions: Synchronization via $\otimes$

$$\dfrac{\dfrac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) :: x : A \otimes B} \qquad \dfrac{\Delta_3, y : A, x : B \vdash R :: z : C}{\Delta_3, x : A \otimes B \vdash x(y).R :: z : C}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big((\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) \mid x(y).R\Big) :: z : C}$$

$$\longrightarrow$$

Principal Cut Reductions: Synchronization via $\otimes$

$$\dfrac{\dfrac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) :: x : A \otimes B} \qquad \dfrac{\Delta_3, y : A, x : B \vdash R :: z : C}{\Delta_3, x : A \otimes B \vdash x(y).R :: z : C}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big((\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) \mid x(y).R\Big) :: z : C}$$

$$\longrightarrow$$

$$\dfrac{\Delta_1 \vdash P :: y : A \qquad \Delta_3, y : A, x : B \vdash R :: z : C}{\Delta_1, x : B, \Delta_3 \vdash (\boldsymbol{\nu} y)(P \mid R) :: z : C}$$

Principal Cut Reductions: Synchronization via $\otimes$

$$\dfrac{\dfrac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) :: x : A \otimes B} \qquad \dfrac{\Delta_3, y : A, x : B \vdash R :: z : C}{\Delta_3, x : A \otimes B \vdash x(y).R :: z : C}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big((\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) \mid x(y).R\Big) :: z : C}$$

$$\longrightarrow$$

$$\dfrac{\Delta_2 \vdash Q :: x : B \qquad \dfrac{\Delta_1 \vdash P :: y : A \qquad \Delta_3, y : A, x : B \vdash R :: z : C}{\Delta_1, x : B, \Delta_3 \vdash (\boldsymbol{\nu} y)(P \mid R) :: z : C}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big(Q \mid (\boldsymbol{\nu} y)(P \mid R)\Big) :: z : C}$$

This way, we have **extracted** the following reduction on processes:

$$(\boldsymbol{\nu} x)\Big((\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) \mid x(y).R\Big) \longrightarrow (\boldsymbol{\nu} x)\Big(Q \mid (\boldsymbol{\nu} y)(P \mid R)\Big)$$

Principal Cut Reductions: Synchronization via $\multimap$

The case of $\multimap$ is similar. We have the following:

$$\dfrac{\dfrac{\Delta_1, y : A \vdash R :: x : B}{\Delta_1 \vdash x(y).R :: x : A \multimap B} \qquad \dfrac{\Delta_2 \vdash P :: y : A \qquad \Delta_3, x : B \vdash Q :: z : C}{\Delta_2, \Delta_3, x : A \multimap B \vdash (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) :: z : C}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big(x(y).R \mid (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q)\Big) :: z : C}$$

which, omitting large bits of the derivation, can be simplified into

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(P \mid R) \mid Q) :: z : C$$

Principal Cut Reductions: Synchronization via $\multimap$

The case of $\multimap$ is similar. We have the following:

$$\dfrac{\dfrac{\Delta_1, y : A \vdash R :: x : B}{\Delta_1 \vdash x(y).R :: x : A \multimap B} \qquad \dfrac{\Delta_2 \vdash P :: y : A \qquad \Delta_3, x : B \vdash Q :: z : C}{\Delta_2, \Delta_3, x : A \multimap B \vdash (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q) :: z : C}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big(x(y).R \mid (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q)\Big) :: z : C}$$

which, omitting large bits of the derivation, can be simplified into

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(P \mid R) \mid Q) :: z : C$$

That is, we have the following reduction on processes:

$$(\boldsymbol{\nu} x)\Big(x(y).R \mid (\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q)\Big) \longrightarrow (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(P \mid R) \mid Q)$$

Where is My Communication?

- In our rules, we have seen processes such as

$$(\nu x)(x(y).R \mid (\nu w)\overline{x}\langle y\rangle.(P \mid Q))$$

where the input parameter and the sent message are the same (i.e. y).

Where is My Communication?

- In our rules, we have seen processes such as

$$(\boldsymbol{\nu} x)(x(y).R \mid (\boldsymbol{\nu} w)\overline{x}\langle y\rangle.(P \mid Q))$$

  where the input parameter and the sent message are the same (i.e. y).

- In general, these names need not match. In that case, reduction involves a name substitution and scope extrusion. In the case of $\multimap$:

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big(x(y).R \mid (\boldsymbol{\nu} w)\overline{x}\langle w\rangle.(P \mid Q)\Big) :: z : C$$

$$\longrightarrow$$

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)((\boldsymbol{\nu} w)(P \mid R\{{}^{w}/_{y}\}) \mid Q) :: z : C$$

Where is My Communication?

- In our rules, we have seen processes such as

$$(\boldsymbol{\nu} x)(x(y).R \mid (\boldsymbol{\nu} w)\overline{x}\langle y\rangle.(P \mid Q))$$

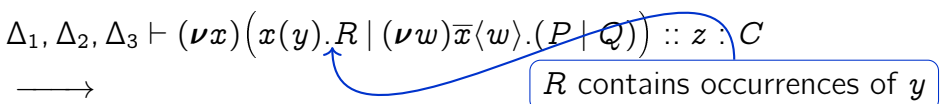
  where the input parameter and the sent message are the same (i.e. y).

- In general, these names need not match. In that case, reduction involves a name substitution and scope extrusion. In the case of $\multimap$:

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\big(x(y).R \mid (\boldsymbol{\nu} w)\overline{x}\langle w\rangle.(P \mid Q)\big) :: z : C$$

$$\longrightarrow$$

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)((\boldsymbol{\nu} w)(P \mid R\{^w/_y\}) \mid Q) :: z : C$$



Where is My Communication?

- In our rules, we have seen processes such as

$$(\boldsymbol{\nu} x)(x(y).R \mid (\boldsymbol{\nu} w)\overline{x}\langle y\rangle.(P \mid Q))$$

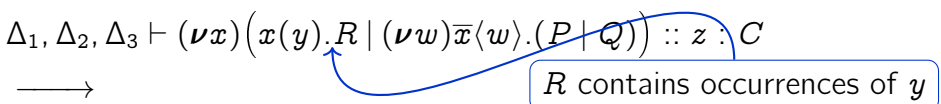
  where the input parameter and the sent message are the same (i.e. y).

- In general, these names need not match. In that case, reduction involves a name substitution and scope extrusion. In the case of $\multimap$:

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\big(x(y).R \mid (\boldsymbol{\nu} w)\overline{x}\langle w\rangle.(P \mid Q)\big) :: z : C$$

$$\longrightarrow$$

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)((\boldsymbol{\nu} w)(P \mid R\{^w/_y\}) \mid Q) :: z : C$$



Where is My Communication?

- In our rules, we have seen processes such as

$$(\boldsymbol{\nu} x)(x(y).R \mid (\boldsymbol{\nu} w)\overline{x}\langle y\rangle.(P \mid Q))$$

  where the input parameter and the sent message are the same (i.e. y).

- In general, these names need not match. In that case, reduction involves a name substitution and scope extrusion. In the case of $\multimap$:

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)\Big(x(y).R \mid (\boldsymbol{\nu} w)\overline{x}\langle w\rangle.(P \mid Q)\Big) :: z : C$$

$$\longrightarrow$$

$$\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)((\boldsymbol{\nu} w)(P \mid R\{^{w}/_{y}\}) \mid Q) :: z : C$$

The scope of w has expanded!

Name w has been received by R

Process Reductions from Cut Reductions: Choice (1/3)

$$\dfrac{\dfrac{\Delta_1 \vdash P :: x : A \qquad \Delta_1 \vdash Q :: x : A}{\Delta_1 \vdash x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} :: x : A \,\&\, B} \qquad \dfrac{\Delta_2, x : A \vdash R :: z : C}{\Delta_2, x : A \,\&\, B \vdash x \triangleleft \mathtt{inl}; R :: z : C}}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} x)\Big(x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} \mid x \triangleleft \mathtt{inl}; R\Big) :: z : C}$$

$$\longrightarrow$$

Process Reductions from Cut Reductions: Choice (2/3)

$$\dfrac{\dfrac{\Delta_1 \vdash P :: x : A \qquad \Delta_1 \vdash Q :: x : A}{\Delta_1 \vdash x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} :: x : A \,\&\, B} \qquad \dfrac{\Delta_2, x : A \vdash R :: z : C}{\Delta_2, x : A \,\&\, B \vdash x \triangleleft \mathtt{inl}; R :: z : C}}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} x)\Big(x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} \mid x \triangleleft \mathtt{inl}; R\Big) :: z : C}$$

$$\longrightarrow$$

$$\dfrac{\Delta_1 \vdash P :: x : A \qquad \Delta_2, x : A \vdash R :: z : C}{}$$

Process Reductions from Cut Reductions: Choice (3/3)

$$\dfrac{\dfrac{\Delta_1 \vdash P :: x : A \qquad \Delta_1 \vdash Q :: x : A}{\Delta_1 \vdash x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} :: x : A \,\&\, B} \qquad \dfrac{\Delta_2, x : A \vdash R :: z : C}{\Delta_2, x : A \,\&\, B \vdash x \triangleleft \mathtt{inl}; R :: z : C}}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} x)\Big(x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} \mid x \triangleleft \mathtt{inl}; R\Big) :: z : C}$$

$$\longrightarrow$$

$$\dfrac{\Delta_1 \vdash P :: x : A \qquad \Delta_2, x : A \vdash R :: z : C}{\Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} x)(P \mid R)}$$

This way, we have extracted the following reduction on processes:

$$(\boldsymbol{\nu} x)\Big(x \triangleright \{\mathtt{inl} : P, \mathtt{inr} : Q\} \mid x \triangleleft \mathtt{inl}; R\Big) \longrightarrow (\boldsymbol{\nu} x)(P \mid R)$$

Process Reductions: Composing Cut and Identity

$$\frac{\overline{y : A \vdash [y \leftarrow x] :: x : A} \qquad \Delta, x : A \vdash P :: z : C}{\Delta \vdash (\boldsymbol{\nu} x)\Big([y \leftarrow x] \mid P\Big) :: z : C}$$

$$\longrightarrow$$

Process Reductions: Composing Cut and Identity

$$\frac{\overline{y : A \vdash [y \leftarrow x] :: x : A} \qquad \Delta, x : A \vdash P :: z : C}{\Delta \vdash (\boldsymbol{\nu} x)\big([y \leftarrow x] \mid P\big) :: z : C}$$

$$\longrightarrow$$

$$\Delta, y : A \vdash P\{y/x\} :: z : C$$

Process Reductions: Composing Cut and Identity

$$\frac{\overline{y : A \vdash [y \leftarrow x] :: x : A} \qquad \Delta, x : A \vdash P :: z : C}{\Delta \vdash (\boldsymbol{\nu} x)\Big([y \leftarrow x] \mid P\Big) :: z : C}$$

$$\longrightarrow$$

$$\Delta, y : A \vdash P\{y/x\} :: z : C$$

This way, we have extracted the following reduction on processes:

$$(\boldsymbol{\nu} x)\Big([y \leftarrow x] \mid P\Big) \longrightarrow P\{y/x\} \qquad (\text{with } y \notin \mathsf{fn}(P))$$

Semantics of Session-typed Processes: Summary

The relation $\longrightarrow$ on processes is defined via principal cut reductions, as follows:

$$
\begin{array}{rcl}
(\boldsymbol{\nu} x)\Big((\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P_1 \mid P_2) \mid x(z).Q\Big) & \longrightarrow & (\boldsymbol{\nu} x)\Big(P_2 \mid (\boldsymbol{\nu} y)(P_1 \mid Q\{{}^{y}/_{z}\})\Big) \\
(\boldsymbol{\nu} x)\Big(x \triangleright \{\mathtt{l_i} : P_i\}_{i\in I} \mid x \triangleleft \mathtt{l_i}; R\Big) & \longrightarrow & (\boldsymbol{\nu} x)\Big(P_i \mid R\Big) \\
(\boldsymbol{\nu} x)\Big(x \triangleleft \mathtt{l_i}; R \mid x \triangleright \{\mathtt{l_i} : P_i\}_{i\in I}\Big) & \longrightarrow & (\boldsymbol{\nu} x)\Big(R \mid P_i\Big) \\
(\boldsymbol{\nu} x)([y \leftarrow x] \mid P) & \longrightarrow & P\{{}^{y}/_{x}\} \qquad (y \text{ is not free in } P) \\
P \longrightarrow P' & \Longrightarrow & (\boldsymbol{\nu} x)(P \mid Q) \longrightarrow (\boldsymbol{\nu} x)(P' \mid Q)
\end{array}
$$

All reductions remove a cut, possibly introducing new (smaller) cuts in the process.

Other Proof Transformations

- **Structural congruence**, noted $\equiv$, concerns "expected" properties of processes. Examples (omitting side conditions):

$$(\nu x)((\nu y)(P \mid Q) \mid R) \equiv (\nu y)(P \mid (\nu x)(Q \mid R))$$
$$(\nu x)(P \mid (\nu y)(Q \mid R)) \equiv (\nu y)(Q \mid (\nu x)(P \mid R))$$

Other Proof Transformations

- **Structural congruence**, noted $\equiv$, concerns "expected" properties of processes. Examples (omitting side conditions):

$$(\nu x)((\nu y)(P \mid Q) \mid R) \equiv (\nu y)(P \mid (\nu x)(Q \mid R))$$
$$(\nu x)(P \mid (\nu y)(Q \mid R)) \equiv (\nu y)(Q \mid (\nu x)(P \mid R))$$

- **Commuting conversions** induce "unusual" process equalities, denoted $\sim_{cc}$. Examples (omitting side conditions):

$$x(u).y(w).P \sim_{cc} y(w).x(u).P$$
$$x(u).(\nu w)\overline{y}\langle w\rangle.(P_1 \mid P_2) \sim_{cc} (\nu w)\overline{y}\langle w\rangle.(x(u).P_1 \mid P_2)$$
$$x(u).(\nu y)(P_1 \mid P_2) \sim_{cc} (\nu y)(x(u).P_1 \mid P_2)$$

  $\sim_{cc}$ can be justfied via a (typed) bisimilarity on processes.

Example: Commuting Conversions

The following proofs are valid: they concern actions on **independent sessions**.

$$\dfrac{\dfrac{\Delta, y : A_1, x : A_2, w : B_1, y : B_2 \vdash P :: z : C}{\Delta, y : A_1, x : A_2, y : B_1 \otimes B_2 \vdash y(w).P :: z : C}}{\Delta, x : A_1 \otimes A_2, y : B_1 \otimes B_2 \vdash x(u).y(w).P :: z : C}$$

We also have:

$$\dfrac{\dfrac{\Delta, y : A_1, x : A_2, w : B_1, y : B_2 \vdash P :: z : C}{\Delta, w : B_1, y : B_2, x : A_1 \otimes A_2 \vdash x(u).P :: z : C}}{\Delta, x : A_1 \otimes A_2, y : B_1 \otimes B_2 \vdash y(w).x(u).P :: z : C}$$

Example: Commuting Conversions

The following proofs are valid: they concern actions on **independent sessions**.

$$\dfrac{\dfrac{\Delta, y : A_1, x : A_2, w : B_1, y : B_2 \vdash P :: z : C}{\Delta, y : A_1, x : A_2, y : B_1 \otimes B_2 \vdash y(w).P :: z : C}}{\Delta, x : A_1 \otimes A_2, y : B_1 \otimes B_2 \vdash x(u).y(w).P :: z : C}$$

We also have:

$$\dfrac{\dfrac{\Delta, y : A_1, x : A_2, w : B_1, y : B_2 \vdash P :: z : C}{\Delta, w : B_1, y : B_2, x : A_1 \otimes A_2 \vdash x(u).P :: z : C}}{\Delta, x : A_1 \otimes A_2, y : B_1 \otimes B_2 \vdash y(w).x(u).P :: z : C}$$

Intuition: The process interpretation is **overly sequential**.

Example: Commuting Conversions (with Cut)

The following proofs are valid: they concern **process optimizations**.

$$\dfrac{\dfrac{\Delta_1, u : A_1, x : A_2 \vdash P_1 :: y : B \qquad \Delta_2, y : B \vdash P_2 :: z : C}{\Delta_1, \Delta_2, u : A_1, x : A_2 \vdash (\boldsymbol{\nu} y)(P_1 \mid P_2) :: z : C}}{\Delta_1, \Delta_2, x : A_1 \otimes A_2 \vdash x(u).(\boldsymbol{\nu} y)(P_1 \mid P_2) :: z : C}$$

We also have:

$$\dfrac{\dfrac{\Delta_1, u : A_1, x : A_2 \vdash P_1 :: y : B}{\Delta_1, x : A_1 \otimes A_2 \vdash x(u).P_1 :: y : B} \qquad \Delta_2, y : B \vdash P_2 :: z : C}{\Delta_1, \Delta_2, x : A_1 \otimes A_2 \vdash (\boldsymbol{\nu} y)(x(u).P_1 \mid P_2) :: z : C}$$

Properties of πDILL

Theorem (Subject Reduction, SR)

If $\Delta \vdash P :: z : C$ *and* $P \longrightarrow Q$ *then* $\Delta \vdash Q :: z : C$

SR ensures our expectations about session fidelity and communication safety.

Properties of πDILL

Theorem (Subject Reduction, SR)

If $\Delta \vdash P :: z : C$ *and* $P \longrightarrow Q$ *then* $\Delta \vdash Q :: z : C$

SR ensures our expectations about session fidelity and communication safety.

Theorem (Deadlock Freedom, DF)

If $\Delta \vdash P :: z : C$ *and* $P \not\longrightarrow$ *then* P *is blocked on either* z *or a channel from* Δ

Corollary

If $\emptyset \vdash P :: z : 1$ *and* $P \not\longrightarrow$ *then* $P = \overline{z}\langle\rangle$.

More on Deadlock Freedom

- Remarkably, the concurrent interpretation of LL leads to a type system that avoids stuck processes.

More on Deadlock Freedom

- Remarkably, the concurrent interpretation of LL leads to a type system that avoids stuck processes.
- A paradigmatic example of a stuck process:

$$P = (\boldsymbol{\nu} x)(\boldsymbol{\nu} z)(x(y).(\boldsymbol{\nu} w)\,\overline{z}\langle w\rangle.(P_1 \mid P_2) \\ \mid (\boldsymbol{\nu} y)z(u).(\overline{x}\langle y\rangle.P_3 \mid P_4))$$

  Note the circular dependency between the two processes in parallel.

More on Deadlock Freedom

- Remarkably, the concurrent interpretation of LL leads to a type system that avoids stuck processes.
- A paradigmatic example of a stuck process:

$$P = (\boldsymbol{\nu} x)(\boldsymbol{\nu} z)(x(y).(\boldsymbol{\nu} w)\,\overline{z}\langle w\rangle.(P_1 \mid P_2) \\ \mid (\boldsymbol{\nu} y) z(u).(\overline{x}\langle y\rangle.P_3 \mid P_4))$$

  Note the circular dependency between the two processes in parallel.

- The following process does not have this dependency:

$$Q = (\boldsymbol{\nu} x)(\boldsymbol{\nu} z)(x(y).(\boldsymbol{\nu} w)\,\overline{z}\langle w\rangle.(Q_1 \mid Q_2) \\ \mid (\boldsymbol{\nu} y)\,\overline{x}\langle y\rangle.(z(u).Q_3 \mid Q_4))$$

  Still, Q cannot be typed in the interpretation - can you see why?

Taking Stock

Today:

- Concurrent interpretation of LL: statics and dynamics
- Left and right rules per connective - rely and guarantee interactive behaviors
- Cut reductions and process synchronizations
- A look at the correctness properties ensured by the logic-based type system
- The computational interpretation of proof transformations
- More on deadlock freedom (DF)

Tomorrow:

- Asynchronous (classical) processes and DF