Propositions as Sessions

Logical Foundations of Concurrent Computation

Dan Frumin and Jorge A. Pérez
University of Groningen, The Netherlands
www.rug.nl/fse/fc
d.frumin | j.a.perez [[at]] rug.nl

ESSLLI 2024
(Part 2, last revised: July 31, 11h)

This Course: Propositions as Sessions

We shall explore the logical foundations of concurrent computation.

Plan:

1. Motivation (Jorge) - Multiplicative, Additive Linear Logic (MAILL) (Dan)

This Course: Propositions as Sessions

We shall explore the logical foundations of concurrent computation.

Plan:

1. Motivation (Jorge) - Multiplicative, Additive Linear Logic (MAILL) (Dan)
2. The concurrent interpretation of MAILL (Jorge)
3. **Today:** Cut-elimination and correctness for concurrent processes (Jorge)
4. Beyond linear resources: the !-modality and resource sharing (Dan)
5. An alternative view of resource sharing: Bunched Implications (Dan)

Your questions and feedback are warmly welcome!

Outline

Preliminaries

Computational Interpretation of LL: Statics
  - Sequent Calculus
  - Output and Input
  - Unit and Axiom
  - Additives
  - Cut
  - Processes

Dynamics
  - Cut Reduction
  - Process Reduction
  - Properties

The Two-Buyer Protocol

Recall the protocol between Alice, Bob, and 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.

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):

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

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):

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

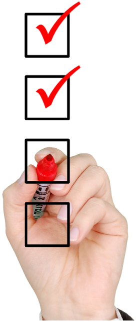
2. A session type for Alice (in its interaction with Bob):

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

Example: A Two-Buyer Protocol

Correctness follows from the interplay of the following properties:

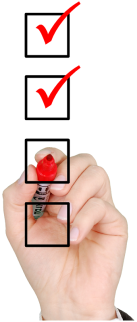
- **Fidelity** – implementations **follow the intended protocol**.
  - Alice never ask Bob twice within the same conversation
  - Alice doesn't continue the transaction if Bob can't contribute
  - Alice chooses among the options provided by Seller



Example: A Two-Buyer Protocol

Correctness follows from the interplay of the following properties:

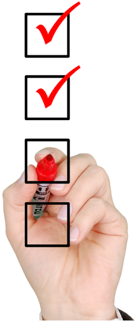
- **Fidelity** – implementations **follow the intended protocol**.
- **Safety** – they don't feature **communication errors**.
    - Seller always returns an integer when Alice requests a quote



Example: A Two-Buyer Protocol

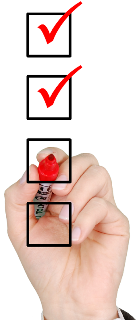
Correctness follows from the interplay of the following properties:

- **Fidelity** – implementations **follow the intended protocol**.
- **Safety** – they don’t feature **communication errors**.
- **Deadlock-Freedom** – they do not “**get stuck**” while running the protocol.
  - Alice eventually receives an answer from Bob on his contribution.



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)

MAILL

$$A, B \quad ::= \quad 1 \mid A \otimes B \mid A \multimap B \mid A \mathbin{\&} B \mid A \oplus B$$

$$A \vdash A \qquad\qquad \emptyset \vdash 1$$

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

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

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

Outline

Preliminaries

Computational Interpretation of LL: Statics
  - Sequent Calculus
  - Output and Input
  - Unit and Axiom
  - Additives
  - Cut
  - Processes

Dynamics
  - Cut Reduction
  - Process Reduction
  - Properties

Propositions As Types

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

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

Propositions As Sessions

Today, the concurrent case:

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

Propositions As Sessions

Today, the concurrent case:

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

We shall follow the correspondence between session types and intuitionistic linear logic (aka πDILL, Caires & Pfenning 2010).

Linear Logic: Sequent Calculus

The sequent

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

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

Linear Logic: Sequent Calculus

The sequent

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

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

Structural rules:

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

Notice: Each connective is "explained" in sequent calculus with a left rule, a right rule, and the interactions with the cut rule.

Linear Logic: Sequent Calculus

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

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

$$\emptyset \vdash 1 \qquad \frac{\Delta \vdash C}{\Delta, 1 \vdash C}$$

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:

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

Process P offers session 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 session C on channel z...

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

Interpreting LL 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	??

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

Interpreting LL 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	??
$\mathtt{end}$	**terminate** the session	??

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

Interpreting LL 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	$U \multimap S$
$\texttt{end}$	**terminate** the session	??

Interpreting LL 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	$U \multimap S$
$\texttt{end}$	**terminate** the session	1

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: $A \otimes B$

Execute P and Q in parallel

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

Send y over x

Propositions as Session Types: $A \otimes B$

Execute P and Q in parallel

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

Declare channel y as **local**, i.e., private

Send y over x

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 ‘⊗’, 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:

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

Propositions as Session Types: The Unit $\mathbf{1}$ and Axiom

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

Propositions as Session Types: The Unit 1 and Axiom

Close the channel x

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

Propositions as Session Types: The Unit 1 and Axiom

Close the channel x

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

Wait for the channel x to close

Propositions as Session Types: The Unit 1 and Axiom

Close the channel x

$$\frac{}{\emptyset \vdash \overline{x}\langle\rangle :: x : 1}$$

$$\frac{\Delta \vdash Q :: z : C}{\Delta, x : 1 \vdash x().Q :: z : C}$$

$$x : A \vdash [y \leftarrow x] :: y : A$$

Wait for the channel x to close

Forward all messages between x and y

Propositions as Session Types: An Alternative for Unit $\mathbf{1}$

We have just seen an explicit interpretation of $\mathbf{1}$. There is also a so-called **silent** interpretation:

$\mathbf{0}$ is the process that does nothing

$$\frac{}{\emptyset \vdash \mathbf{0} :: x : \mathbf{1}} \qquad \frac{\Delta \vdash Q :: z : C}{\Delta, x : \mathbf{1} \vdash Q :: z : C}$$

No explicit action

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	$U \multimap S$

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

$\mathtt{end}$	**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
$S_1 \mathbin{\&} S_2$	**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:

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

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

Notice: ‘◁’ means sending a label and ‘▷’ means receiving a label.

Propositions as Session Types: Additive Conjunction

The generalization to n-ary operators:

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

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

Propositions as Session Types: Additive Conjunction

Let's examine first at the binary version:

Branch on x: proceed either as P or Q

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

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

Propositions as Session Types: Additive Conjunction

Let's examine first at the binary version:

Branch on x: proceed either as P or Q

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

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

Select either left or right session continuation

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

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

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

Select exactly one of the session continuations

Propositions as Session Types: Additive Disjunction

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

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

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

Propositions as Session Types: Additive Disjunction

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

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

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

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

A variant of the **π-calculus** (Milner, Parrow & Walker, 1992):

$P, Q ::=$	$[y \leftarrow x]$	**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)

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

P can provide A along x

Q relies on x along A to provide $z : C$

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

Propositions as Session Types: Cut

P can provide A along x

Q relies on x along A to provide $z : C$

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

Propositions as Session Types: Cut

P can provide A along x

Q relies on x along A to provide $z : C$

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

Execute P and Q in parallel

Propositions as Session Types: Cut

P can provide A along x

Q relies on x along A to provide $z : C$

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

Execute P and Q in parallel

Declare channel x **local** to P and Q

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

- 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.

Open and Closed Systems

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

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

  then we can say P is an **open** system: it is ready to be composed with other processes (via “interfaces” $x_1, \ldots , x_n$).

Open and Closed Systems

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

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

  then we can say P is an **open** system: it is ready to be composed with other processes (via “interfaces” $x_1, \ldots, x_n$).
- On the other hand, a process such as $\emptyset \vdash Q :: y : A$ is a **closed** system: it has only one visible “interface”, namely y.

Open and Closed Systems

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

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

  then we can say P is an **open** system: it is ready to be composed with other processes (via "interfaces" $x_1, \ldots, x_n$).
- On the other hand, a process such as $\emptyset \vdash Q :: y : A$ is a **closed** system: it has only one visible "interface", namely y.
- The interpretation of cut as typed composition gives an immediate recipe for closing systems. Example:

Open and Closed Systems

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

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

  then we can say P is an **open** system: it is ready to be composed with other processes (via "interfaces" $x_1, \ldots, x_n$).
- On the other hand, a process such as $\emptyset \vdash Q :: y : A$ is a **closed** system: it has only one visible "interface", namely y.
- The interpretation of cut as typed composition gives an immediate recipe for closing systems. 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

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

  then we can say P is an **open** system: it is ready to be composed with other processes (via "interfaces" $x_1, \ldots, x_n$).
- On the other hand, a process such as $\emptyset \vdash Q :: y : A$ is a **closed** system: it has only one visible "interface", namely y.
- The interpretation of cut as typed composition gives an immediate recipe for closing systems. 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

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

  then we can say P is an **open** system: it is ready to be composed with other processes (via "interfaces" $x_1, \ldots, x_n$).
- On the other hand, a process such as $\emptyset \vdash Q :: y : A$ is a **closed** system: it has only one visible "interface", namely y.
- The interpretation of cut as typed composition gives an immediate recipe for closing systems. 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

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

  then we can say P is an **open** system: it is ready to be composed with other processes (via “interfaces” $x_1, \ldots, x_n$).
- On the other hand, a process such as $\emptyset \vdash Q :: y : A$ is a **closed** system: it has only one visible “interface”, namely y.
- The interpretation of cut as typed composition gives an immediate recipe for closing systems. 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.

Open and Closed Systems

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

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

  then we can say P is an **open** system: it is ready to be composed with other processes (via "interfaces" $x_1, \ldots, x_n$).
- On the other hand, a process such as $\emptyset \vdash Q :: y : A$ is a **closed** system: it has only one visible "interface", namely y.
- The interpretation of cut as typed composition gives an immediate recipe for closing systems. 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.
- 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.

Processes for The Two-Buyer Protocol

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

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

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

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

(This is the silent interpretation!)

Processes for The Two-Buyer Protocol

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

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

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

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

  (This is the silent interpretation!)

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

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

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

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:

$$b : \mathrm{bobProto}, s : \mathrm{sellerProto} \vdash \textsf{Alice} :: z : 1$$

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

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

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

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

Outline

- Preliminaries
- Computational Interpretation of LL: Statics
  - Sequent Calculus
  - Output and Input
  - Unit and Axiom
  - Additives
  - Cut
  - Processes
- Dynamics
  - Cut Reduction
  - Process Reduction
  - Properties

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 still need to see 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)
- 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 $\otimes$ (1/4)

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

$$\longrightarrow$$

Principal Cut Reductions: Synchronization via $\otimes$ (2/4)

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

$$\longrightarrow$$

Principal Cut Reductions: Synchronization via $\otimes$ (3/4)

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

$$\longrightarrow$$

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

Principal Cut Reductions: Synchronization via $\otimes$ (4/4)

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

$$\longrightarrow$$

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

This way, we have 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

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

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, the name (on x) and the input parameter are the same (i.e. y).

In general, these names need not match and 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, the name (on x) and the input parameter are the same (i.e. y).

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

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

$$\longrightarrow$$

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

R contains occurrences of y

Where is My Communication?

In our rules, the name (on x) and the input parameter are the same (i.e. y).

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

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

$$\longrightarrow$$

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

R contains occurrences of y

Name w has been received by R

Where is My Communication?

In our rules, the name (on x) and the input parameter are the same (i.e. y).

In general, these names need not match and 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)

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

$$\longrightarrow$$

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

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

$$\longrightarrow$$

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

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

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

$$\longrightarrow$$

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

This way, we have the following reduction on processes:

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

Semantics of Session-typed Processes

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

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

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

Other Proof Transformations

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

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

Other Proof Transformations

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

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

- **Commuting conversions** induce “peculiar” equalities on processes, denoted $\sim_{cc}$. Examples (omitting side conditions):

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

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

Properties of πDILL

Theorem (Subject reduction)

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

SR ensures our (informal) expectations about session fidelity and communication safety.

Properties of πDILL

Theorem (Subject reduction)

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

SR ensures our (informal) expectations about session fidelity and communication safety.

Theorem (Deadlock-freedom)

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 / Progress

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

More on Deadlock-Freedom / Progress

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

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

Note the circular dependency between the two processes in parallel.

More on Deadlock-Freedom / Progress

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

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

  Note the circular dependency between the two processes in parallel.
- The following process does not have this dependency:

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

  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 first look at correctness properties ensured by the logic-based type system
- More on the computational interpretation of proof transformations
- Deadlock-freedom and progress

Coming next:

- Beyond linear resources: the !-modality

Propositions as Sessions

Logical Foundations of Concurrent Computation

Dan Frumin and Jorge A. Pérez
University of Groningen, The Netherlands
www.rug.nl/fse/fc
d.frumin | j.a.perez [[at]] rug.nl

ESSLLI 2024
(Part 2, last revised: July 31, 11h)