# 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 30, 2026)

# Unrestricted Behavior: Clients and Servers

# Outline

# 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:

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

- Recall: we write $\overline{x}(y).P$ to stand for $(\nu y)\overline{x}\langle y\rangle.P$.
  In process $\overline{x}(y).P \mid x(z).Q$, the local name $y$ is 'secret' to $P$ (unknown to $Q$).
  We have a synchronization:

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

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

# Process Synchronizations, Informally

- In the 'basic' reduction step the output $\overline{x}\langle y \rangle$ and the input $x(z)$ are **consumed** after the reduction.

# Process Synchronizations, Informally

- In the 'basic' reduction step the output $\overline{x}\langle y \rangle$ and the input $x(z)$ are **consumed** after the reduction.
- The **replicated process** $!P$ represents arbitrarily many parallel copies of $P$:

$$!P \triangleq P \mid P \mid \cdots \mid P$$

- That is, $!P$ can be used zero or many times.
  As we will see, ! is a logic modality with contraction and weakening.

# Process Synchronizations, Informally

- In the 'basic' reduction step the output $\overline{x}\langle y\rangle$ and the input $x(z)$ are **consumed** after the reduction.
- The **replicated process** $!P$ represents arbitrarily many parallel copies of $P$:

$$!P \triangleq P \mid P \mid \cdots \mid P$$

- That is, $!P$ can be used zero or many times.
  As we will see, ! is a logic modality with contraction and weakening.
- When we use ! for input actions, we obtain a **client-server** behavior:

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

  Note: the output $\overline{x}\langle y\rangle$ acts as a client request to the server $!x(z).Q$.
  The server persists after reduction (for the benefit of other clients).

# Process Synchronizations, Informally

- The **replicated process** $!x(z).P$ represents arbitrarily many copies of $x(z).P$:

$$!x(z).P \triangleq x(z).P \mid x(z).P \mid \cdots \mid x(z).P$$

# Process Synchronizations, Informally

- The **replicated process** $!x(z).P$ represents arbitrarily many copies of $x(z).P$:

$$!x(z).P \triangleq x(z).P \mid x(z).P \mid \cdots \mid x(z).P$$

- We can use ! to obtain a **client-server** behavior:

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

Note: the output $\overline{x}\langle y\rangle$ acts as a client request to the server $!x(z).Q$.
The server persists after reduction (for the benefit of other clients).

# Process Synchronizations, Informally

- The **replicated process** $!x(z).P$ represents arbitrarily many copies of $x(z).P$:

$$!x(z).P \triangleq x(z).P \mid x(z).P \mid \cdots \mid x(z).P$$

- We can use ! to obtain a **client-server** behavior:

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

  Note: the output $\overline{x}\langle y\rangle$ acts as a client request to the server $!x(z).Q$.
  The server persists after reduction (for the benefit of other clients).

- When the client sends a **private name**, we then obtain the following:

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

# Process Synchronizations, Informally

- Client-server interactions can easily lead to **non-terminating behaviors**.
- **Example**: Consider following the server

$$S \triangleq !x(y).\overline{x}\langle y\rangle.\overline{y}\langle\rangle$$

The server $S$ sends a signal on the channel $y$ sent by the client, and triggers an additional request (to itself).

# Process Synchronizations, Informally

- Client-server interactions can easily lead to **non-terminating behaviors**.
- **Example**: Consider following the server

$$S \triangleq !x(y).\overline{x}\langle y\rangle.\overline{y}\langle\rangle$$

The server $S$ sends a signal on the channel $y$ sent by the client, and triggers an additional request (to itself). For instance:

$$\begin{aligned} S \mid \overline{x}\langle z\rangle.0 &\longrightarrow S \mid \overline{x}\langle z\rangle.\overline{z}\langle\rangle \\ &\longrightarrow S \mid \overline{z}\langle\rangle \mid \overline{x}\langle z\rangle.\overline{z}\langle\rangle \\ &\longrightarrow S \mid \overline{z}\langle\rangle \mid \overline{z}\langle\rangle \mid \overline{x}\langle z\rangle.\overline{z}\langle\rangle \longrightarrow \cdots \end{aligned}$$

- Type systems can control clients and servers to ensure correctness.

# Propositions as Sessions

| | | |
|---|---|---|
| Linear Logic propositions | $\Longleftrightarrow$ | Session types describing behavior |
| Sequent calculus derivations | $\Longleftrightarrow$ | $\pi$-calculus processes |
| Cut reductions | $\Longleftrightarrow$ | Communication between processes |

# Propositions as Sessions

| | | |
|---|---|---|
| Linear Logic propositions | $\Longleftrightarrow$ | Session types describing behavior |
| Sequent calculus derivations | $\Longleftrightarrow$ | $\pi$-calculus processes |
| Cut reductions | $\Longleftrightarrow$ | Communication between processes |

**Today**: Incorporating **servers** into the framework.

# MAILL: Multiplicative Additive Intuitionistic LL

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

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

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

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

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

# 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:

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

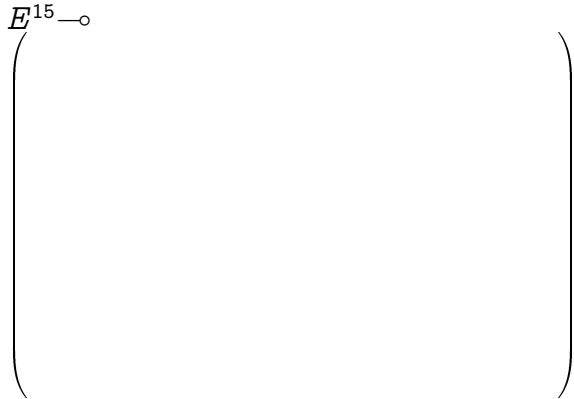
Process $P$ **offers** protocol $C$ on channel $z$...

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

# Outline

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)



# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \Big( (Sal \,\&\, Soup) \otimes \Big)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \left( \begin{array}{l} (Sal \;\&\; Soup) \otimes \\ \Big( (Schn \;\&\; Tofu \;\&\; (E \otimes E \multimap Fish)) \end{array} \right)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \Big( (Sal \,\&\, Soup) \otimes \big( (Schn \,\&\, Tofu \,\&\, (E \otimes E \multimap Fish)) \otimes Fries \big) \otimes \Big)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \left( \begin{array}{l} (Sal \,\&\, Soup) \otimes \\ \Big( (Schn \,\&\, Tofu \,\&\, (E \otimes E \multimap Fish)) \\ \qquad \otimes \; Fries \Big) \otimes \\ (IceCr \,\&\, Cheese) \otimes \end{array} \right)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Menu at the Linear Logic Cafe

**Summer Menu! €15 p.p.**
**Starter**: Greek Salad, or Soup
**Main**: Chicken Schnitzel, Tofu, or Salmon (for extra €2)
(all main dishes come with a side of fries)
**Desert**: Ice Cream, or Cheese Platter
**Drinks:** Leffe Tripel (for extra €3)

$$E^{15} \multimap \left( \begin{array}{l} (Sal \,\&\, Soup) \otimes \\ \Big( (Schn \,\&\, Tofu \,\&\, (E \otimes E \multimap Fish)) \\ \qquad \otimes \; Fries \Big) \otimes \\ (IceCr \,\&\, Cheese) \otimes \\ \\ (E^3 \multimap Beer) \end{array} \right)$$

**Note:** We use & to indicate that the kitchen can provide any option you choose.

# Restricted vs Unrestricted resources

Intuitionistic (and classical) logic:

$$A \wedge (A \to B) \vdash B \wedge A \wedge (A \to B)$$

Linear Logic:

$$A \otimes (A \multimap B) \vdash B$$
$$A \otimes (A \multimap B) \vdash A \otimes (A \multimap B)$$

# Restricted vs Unrestricted resources

Intuitionistic (and classical) logic:

$$A \wedge (A \to B) \vdash B \wedge A \wedge (A \to B)$$

Linear Logic:

$$A \otimes (A \multimap B) \vdash B$$
$$A \otimes (A \multimap B) \vdash A \otimes (A \multimap B)$$

$$A \otimes (A \multimap B) \vdash B \,\&\, (A \otimes (A \multimap B))$$

# Unrestricted Resources

- So far we have only talked about **linear** (restricted) resources.

# Unrestricted Resources

- So far we have only talked about **linear** (restricted) resources.
- A session type $A$ describes a finite, linear session.

# Unrestricted Resources

- So far we have only talked about **linear** (restricted) resources.
- A session type $A$ describes a finite, linear session.
- Composition on disjoint sessions:

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

# Unrestricted Resources

- So far we have only talked about **linear** (restricted) resources.
- A session type $A$ describes a finite, linear session.
- Composition on disjoint sessions:

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

Divide all the resources between $P$ and $Q$

# Unrestricted Resources

- So far we have only talked about **linear** (restricted) resources.
- A session type $A$ describes a finite, linear session.
- Composition on disjoint sessions:

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

- Question: How to introduce **unrestricted** (i.e. non-linear) sessions?

# The ! Modality

$!A$ read: '**of course** $A$', or '**bang** $A$', or '**exponential** $A$'

Some principles:

$$!A \vdash A \qquad !A \vdash !A \otimes !A \qquad !A \vdash 1 \qquad \frac{A \vdash B}{!A \vdash !B}$$

- $!A$ satisfies contraction and weakening.

# The ! Modality

$!A$ read: '**of course** $A$', or '**bang** $A$', or '**exponential** $A$'

Some principles:

$$!A \vdash A \qquad !A \vdash !A \otimes !A \qquad !A \vdash 1 \qquad \frac{A \vdash B}{!A \vdash !B}$$

- $!A$ satisfies contraction and weakening.
- A list-like presentation: $!A \vdash 1 \,\&\, (A \otimes !A)$

# The ! Modality

$!A$ read: '**of course** $A$', or '**bang** $A$', or '**exponential** $A$'

Some principles:

$$!A \vdash A \qquad !A \vdash !A \otimes !A \qquad !A \vdash 1 \qquad \frac{A \vdash B}{!A \vdash !B}$$

- $!A$ satisfies contraction and weakening.
- A list-like presentation: $!A \vdash 1 \,\&\, (A \otimes !A)$
- $A \otimes !(A \multimap B) \vdash B \otimes !(A \multimap B)$

# The ! Modality

Some useful properties of !:

- $!(A \,\&\, B) \dashv\vdash !A \otimes !B$

# The ! Modality

Some useful properties of !:

- $!(A \,\&\, B) \dashv\vdash !A \otimes !B$
- $!(A \otimes B) \dashv\vdash !A \otimes !B$

# The ! Modality

Some useful properties of !:

- $!(A \,\&\, B) \dashv\vdash\, !A \otimes\, !B$
- $!(A \otimes B) \dashv\vdash\, !A \otimes\, !B$
- $!(A \,\&\, B) \vdash\, !A \,\&\, !B$
- $!A \,\&\, !B \nvdash !(A \,\&\, B)$

# The ! Modality

Some useful properties of !:

- $!(A \,\&\, B) \dashv\vdash\, !A \otimes\, !B$
- $!(A \otimes B) \dashv\vdash\, !A \otimes\, !B$
- $!(A \,\&\, B) \vdash\, !A \,\&\, !B$
- $!A \,\&\, !B \nvdash\, !(A \,\&\, B)$
- $!A \dashv\vdash\, !!A$

# The ! Modality as a Session Type

- $!A$: Unlimited copies of the session $A$. That is, a server for session $A$

# The ! Modality as a Session Type

- $!A$: Unlimited copies of the session $A$. That is, a server for session $A$
- We refine our judgment to account for two kinds of contexts:

$$\Gamma; \Delta \vdash P :: z : C$$

Unrestricted resources appear in $\Gamma$, restricted resources appear in $\Delta$.

# The ! Modality as a Session Type

- $!A$: Unlimited copies of the session $A$. That is, a server for session $A$
- We refine our judgment to account for two kinds of contexts:

$$\Gamma; \Delta \vdash P :: z : C$$

  Unrestricted resources appear in $\Gamma$, restricted resources appear in $\Delta$.
- This variant is known as DILL: Dual Intuitionistic Linear Logic.

# The ! Modality as a Session Type

- $!A$: Unlimited copies of the session $A$. That is, a server for session $A$
- We refine our judgment to account for two kinds of contexts:

$$\Gamma; \Delta \vdash P :: z : C$$

  Unrestricted resources appear in $\Gamma$, restricted resources appear in $\Delta$.
- This variant is known as DILL: Dual Intuitionistic Linear Logic.
- The judgment $A_1, \ldots, A_k; B_1, \ldots, B_n \vdash C$ stands for

$$!A_1 \otimes \cdots \otimes !A_k \otimes B_1 \otimes \cdots \otimes B_n \multimap C$$

# The ! Modality as a Session Type

- $!A$: Unlimited copies of the session $A$. That is, a server for session $A$
- We refine our judgment to account for two kinds of contexts:

$$\Gamma; \Delta \vdash P :: z : C$$

  Unrestricted resources appear in $\Gamma$, restricted resources appear in $\Delta$.
- This variant is known as DILL: Dual Intuitionistic Linear Logic.
- The judgment $A_1, \ldots, A_k; B_1, \ldots, B_n \vdash C$ stands for

$$!A_1 \otimes \cdots \otimes !A_k \otimes B_1 \otimes \cdots \otimes B_n \multimap C$$

- Unrestricted resources are non-linear and can be shared. The cut rule becomes:

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

# The ! Modality as a Session Type

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

$$\frac{\Gamma;\ \Delta_1 \vdash P :: y : A \qquad \Gamma;\ \Delta_2 \vdash Q :: x : B}{\Gamma;\ \Delta_1, \Delta_2 \vdash \overline{x}\langle y \rangle.(P \mid Q) :: x : A \otimes B} \qquad \frac{}{\Gamma;\ x : A \vdash [y \leftarrow x] :: y :: A}$$

*...etc*

# The ! Modality as a Session Type

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

$$\frac{\Gamma;\ \Delta_1 \vdash P :: y : A \qquad \Gamma; \Delta_2 \vdash Q :: x : B}{\Gamma;\ \Delta_1, \Delta_2 \vdash \overline{x}\langle y \rangle.(P \mid Q) :: x : A \otimes B} \qquad \frac{}{\Gamma;\ x : A \vdash [y \leftarrow x] :: y :: A}$$

*...etc*

Plus contraction and weakening for unrestricted resources (in $\Gamma$):

$$\frac{x : A, y : A, \Gamma;\ \Delta \vdash P :: z : C}{x : A, \Gamma;\ \Delta \vdash P\{x/y\} :: z : C} \qquad \frac{\Gamma;\ \Delta \vdash P :: z : C}{x : A, \Gamma;\ \Delta \vdash P :: z : C}$$

# The Concurrent Interpretation, Now With Servers

To extend the interpretation, we proceed as before:

- We consider right and left rules for the modality $!A$.
- We annotate the rules with suitable constructs, expressing servers and clients.
- We identify composition principles that enable cut reductions, which can be interpreted as process reductions.

# The Concurrent Interpretation, Now With Servers

To extend the interpretation, we proceed as before:

- We consider right and left rules for the modality $!A$.
- We annotate the rules with suitable constructs, expressing servers and clients.
- We identify composition principles that enable cut reductions, which can be interpreted as process reductions.

In the presence of unrestricted resources, we have an implicit source of client-servers behaviors, given by $\Gamma$.

We can also be explicit via the modality $!A$ at the level of propositions.

# The Concurrent Interpretation, Now With Servers

!-move

$$\frac{u : A, \Gamma;\ y : A, \Delta \vdash Q :: z : C}{u : A, \Gamma;\ \Delta \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: z : C}$$

!-cut

$$\frac{\Gamma;\ \emptyset \vdash P :: y : A \qquad \Gamma, x : A;\ \Delta \vdash Q :: z : C}{\Gamma;\ \Delta \vdash (\boldsymbol{\nu} x)(!x(y).P \mid Q) :: z : C}$$

!R

$$\frac{\Gamma;\ \emptyset \vdash P :: y : A}{\Gamma;\ \emptyset \vdash\ !u(y).P :: u : !A}$$

!L

$$\frac{u : A, \Gamma;\ \Delta \vdash P :: z : C}{\Gamma;\ \Delta, x : !A \vdash P\{^{x}/_{u}\} :: z : C}$$

With the following reduction rule:

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

# The ! Modality: Servers

The right rule for !:

$$\frac{\Gamma; \emptyset \vdash A}{\Gamma; \emptyset \vdash\ !A}$$

- We can derive $!A$ from $A$ if the proof of $A$ (i.e., a process) does **not** depend on linear/restricted resources.

# The ! Modality: Servers

The right rule for !:

$$\frac{\Gamma; \emptyset \vdash P :: y : A}{\Gamma; \emptyset \vdash\ !u(y).P :: u : !A}$$

- We can derive $!A$ from $A$ if the proof of $A$ (i.e., a process) does **not** depend on linear/restricted resources.
- The server $!u(y).P$ is always ready to receive a request (with a name) on $u$, and to provide a copy of the body $P$ for that request.

# The ! Modality: Clients

$$\frac{u : A, \Gamma; \Delta \vdash C}{\Gamma; x : !A, \Delta \vdash C} \qquad \frac{u : A, \Gamma; y : A, \Delta \vdash C}{u : A, \Gamma; \Delta \vdash C}$$

# The ! Modality: Clients

$$\frac{u : A, \Gamma; \Delta \vdash P :: z : C}{\Gamma; x : !A, \Delta \vdash P\{x/u\} :: z : C} \qquad \frac{u : A, \Gamma; y : A, \Delta \vdash \quad C}{u : A, \Gamma; \Delta \vdash \quad C}$$

- We can move $!A$ from the restricted to the unrestricted section.

# The ! Modality: Clients

$$\frac{u : A, \Gamma; \Delta \vdash P :: z : C}{\Gamma; x : !A, \Delta \vdash P\{x/u\} :: z : C} \qquad \frac{u : A, \Gamma; y : A, \Delta \vdash Q :: z : C}{u : A, \Gamma; \Delta \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: z : C}$$

- We can move $!A$ from the restricted to the unrestricted section.
- A client can request a copy of $A$ by creating a new name and sending the request to $!A$.

# Client-Server Interaction

We expect to obtain the following reduction rule:

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

# Client-Server Interaction

We expect to obtain the following reduction rule:

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

For convenience, we will also use the **derived composition rule** for servers:

$$\text{!-cut}\quad \frac{\Gamma;\, \emptyset \vdash P :: y : A \qquad \Gamma, x : A;\, \Delta \vdash Q :: z : C}{\Gamma;\, \Delta \vdash (\boldsymbol{\nu} x)(!x(y).P \mid Q) :: z : C}$$

# The ! Modality: Client-Server Interaction (1)

The interaction between ‘move’ and ‘!-cut’ is simplified using a ‘!-cut’ and ‘cut’:

$$\dfrac{\Gamma;\, \emptyset \vdash P :: x : A \qquad \dfrac{\Gamma, u : A;\, \Delta, y : A \vdash Q :: z : C}{\Gamma, u : A;\, \Delta \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: z : C}}{\Gamma;\, \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q) :: z : C}$$

$$\longrightarrow$$

# The ! Modality: Client-Server Interaction (1)

The interaction between ‘move’ and ‘!-cut’ is simplified using a ‘!-cut’ and ‘cut’:

$$\dfrac{\Gamma;\, \emptyset \vdash P :: x : A \qquad \dfrac{\Gamma, u : A;\, \Delta, y : A \vdash Q :: z : C}{\Gamma, u : A;\, \Delta \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: z : C}}{\Gamma;\, \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q) :: z : C}$$

$$\longrightarrow$$

$$\dfrac{\Gamma;\, \emptyset \vdash P :: x : A \qquad \Gamma, u : A;\, \Delta, y : A \vdash Q :: z : C}{}$$

# The ! Modality: Client-Server Interaction (1)

The interaction between ‘move’ and ‘!-cut’ is simplified using a ‘!-cut’ and ‘cut’:

$$\dfrac{\Gamma;\, \emptyset \vdash P :: x : A \qquad \dfrac{\Gamma, u : A;\, \Delta, y : A \vdash Q :: z : C}{\Gamma, u : A;\, \Delta \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: z : C}}{\Gamma;\, \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q) :: z : C}$$

$$\longrightarrow$$

$$\dfrac{\Gamma;\, \emptyset \vdash P :: x : A \qquad \dfrac{\Gamma;\, \emptyset \vdash P :: x : A \qquad \Gamma, u : A;\, \Delta, y : A \vdash Q :: z : C}{\Gamma;\, \Delta, y : A \vdash (\boldsymbol{\nu} u)(!u(x).P \mid Q) :: z : C}}{\Gamma;\, \Delta \vdash (\boldsymbol{\nu} u)(P\{{}^{y}/_{x}\} \mid (\boldsymbol{\nu} u)(!u(x).P \mid Q)) :: z : C}$$

# The ! Modality: Client-Server Interaction (2)

Now the interaction between the right/left rules for $!A$. Write $T$ instead of $z : C$:

$$\dfrac{\dfrac{\Gamma;\emptyset \vdash P :: x : A}{\Gamma;\emptyset \vdash !u(x).P :: u : !A} \qquad \dfrac{\dfrac{\Gamma, u : A;\, \Delta, y : A \vdash Q :: T}{\Gamma, u : A;\, \Delta \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: T}}{\Gamma;\, \Delta, u : !A \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: T}}{\Gamma;\, \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q) :: T}$$

$$\longrightarrow$$

# The ! Modality: Client-Server Interaction (2)

Now the interaction between the right/left rules for $!A$. Write $T$ instead of $z : C$:

$$\dfrac{\dfrac{\Gamma;\, \emptyset \vdash P :: x : A}{\Gamma;\, \emptyset \vdash !u(x).P :: u : !A} \qquad \dfrac{\dfrac{\Gamma, u : A;\, \Delta, y : A \vdash Q :: T}{\Gamma, u : A;\, \Delta \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: T}}{\Gamma;\, \Delta, u : !A \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: T}}{\Gamma;\, \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q) :: T}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\Gamma;\, \emptyset \vdash P :: x : A}{\Gamma, u : A;\, \emptyset \vdash P\{y/x\} :: y : A} \qquad \Gamma, u : A;\, \Delta, y : A \vdash Q :: T}{\Gamma, u : A;\, \Delta \vdash (\boldsymbol{\nu} y)(P\{y/x\} \mid Q) :: T}$$

# The ! Modality: Client-Server Interaction (2)

Now the interaction between the right/left rules for $!A$. Write $T$ instead of $z : C$:

$$\dfrac{\dfrac{\Gamma;\emptyset \vdash P :: x : A}{\Gamma;\emptyset \vdash !u(x).P :: u : !A} \qquad \dfrac{\dfrac{\Gamma, u : A;\ \Delta, y : A \vdash Q :: T}{\Gamma, u : A;\ \Delta \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: T}}{\Gamma;\ \Delta, u : !A \vdash (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q :: T}}{\Gamma;\ \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid (\boldsymbol{\nu} y)\overline{u}\langle y\rangle.Q) :: T}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\Gamma;\emptyset \vdash P :: x : A}{\Gamma, u : A;\emptyset \vdash P\{y/x\} :: y : A} \qquad \dfrac{\dfrac{\Gamma, u : A;\ \Delta, y : A \vdash Q :: T}{\Gamma, u : A;\ \Delta \vdash (\boldsymbol{\nu} y)(P\{y/x\} \mid Q) :: T}}{\Gamma;\ \Delta, u : !A \vdash (\boldsymbol{\nu} x)(P\{y/x\} \mid Q) :: T}}{\Gamma;\ \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid (\boldsymbol{\nu} y)(P\{y/x\} \mid Q)) :: T}$$

# List-like Specification for Servers

$$\frac{u : A;\ \emptyset \vdash \ldots :: w : 1 \,\&\, (A \otimes\, !A)}{\emptyset;\ u : !A \vdash \ldots :: w : 1 \,\&\, (A \otimes\, !A)}$$

# List-like Specification for Servers

$$\dfrac{\dfrac{u : A;\, \emptyset \vdash \ldots :: w : 1 \qquad \overline{u : A;\, \emptyset \vdash \ldots :: w : A \otimes\, !A}}{u : A;\, \emptyset \vdash w \triangleright \left\{ \begin{array}{l} \texttt{inl} : \ldots, \\ \texttt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes\, !A)}}{\dfrac{u : A;\, \emptyset \vdash w \triangleright \left\{ \begin{array}{l} \texttt{inl} : \ldots, \\ \texttt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes\, !A)}{\emptyset;\, u : !A \vdash w \triangleright \left\{ \begin{array}{l} \texttt{inl} : \ldots, \\ \texttt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes\, !A)}}$$

# List-like Specification for Servers

$$\dfrac{\dfrac{\dfrac{u : A;\ \emptyset \vdash \overline{w}\langle\rangle :: w : 1 \qquad \overline{u : A;\ \emptyset \vdash \ldots :: w : A \otimes\, !A}}{u : A;\ \emptyset \vdash w \triangleright \left\{ \begin{array}{l} \mathtt{inl} : \overline{w}\langle\rangle, \\ \mathtt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes\, !A)}}{u : A;\ \emptyset \vdash w \triangleright \left\{ \begin{array}{l} \mathtt{inl} : \overline{w}\langle\rangle, \\ \mathtt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes\, !A)}}{\emptyset;\ u : !A \vdash w \triangleright \left\{ \begin{array}{l} \mathtt{inl} : \overline{w}\langle\rangle, \\ \mathtt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes\, !A)}$$

# List-like Specification for Servers

$$\dfrac{\dfrac{u : A;\ \emptyset \vdash \overline{u}(x).[y \leftarrow x] :: y : A \qquad u : A;\ \emptyset \vdash [w \leftarrow u] :: w :: !A}{u : A;\ \emptyset \vdash \overline{w}(y).(\overline{u}(x).[y \leftarrow x]) \mid [w \leftarrow u]) :: w : A \otimes !A} \qquad u : A;\ \emptyset \vdash \overline{w}\langle\rangle :: w : 1}{u : A;\ \emptyset \vdash w \triangleright \left\{ \begin{array}{l} \texttt{inl} : \overline{w}\langle\rangle, \\ \texttt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes !A)}$$

$$\dfrac{u : A;\ \emptyset \vdash w \triangleright \left\{ \begin{array}{l} \texttt{inl} : \overline{w}\langle\rangle, \\ \texttt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes !A)}{\emptyset;\ u : !A \vdash w \triangleright \left\{ \begin{array}{l} \texttt{inl} : \overline{w}\langle\rangle, \\ \texttt{inr} : \ldots \end{array} \right\} :: w : 1 \,\&\, (A \otimes !A)}$$

# Bar Tending at the Linear Logic Cafe

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset;\ e : E \otimes E \otimes E \vdash depo(e,x) :: x : 1}\qquad \dfrac{\dfrac{\dfrac{}{\emptyset;\ k : Keg \vdash pour(k,b) :: b : Beer}}{u : Keg;\ \emptyset \vdash \overline{u}(k).pour(k,b) :: b : Beer}}{u : Keg;\ x : 1 \vdash x().\overline{u}(k).pour(k,b) :: b : Beer}}{u : Keg;\ e : E^3 \vdash (\boldsymbol{\nu} x)\big(depo(e,x) \mid x().\overline{u}(k).pour(k,b)\big) :: b : Beer}}{u : Keg;\ \emptyset \vdash b(e).(\boldsymbol{\nu} x)\big(depo(e,x) \mid x().\overline{u}(k).pour(k,b)\big) :: b : E^3 \multimap Beer}}{u : Keg;\ \emptyset \vdash !t(b).b(e).(\boldsymbol{\nu} x)\big(depo(e,x) \mid x().\overline{u}(k).pour(k,b)\big) :: t : !(E^3 \multimap Beer)}$$

# Outline

# Clients and Servers in CLL

We have just seen clients and servers in the intuitionistic setting (DILL).

In Wadler's CP, clients and servers appear differently.

- We have exponential modalities: $!A$ for servers, and its dual $?A$ for clients. At the level of judgments, there is no separation between linear and unrestricted resources.

# Clients and Servers in CLL

- There is one rule for servers:

$$\frac{P \vdash ?\Gamma, y : A}{!x(y).P \vdash ?\Gamma, x : !A}\ [!]$$

  $?\Gamma$ ensures: A process may only provide a server if it is implemented by communicating only with other servers.

# Clients and Servers in CLL

- There is one rule for servers:

$$\frac{P \vdash ?\Gamma, y : A}{!x(y).P \vdash ?\Gamma, x : !A}\ [!]$$

  $?\Gamma$ ensures: A process may only provide a server if it is implemented by communicating only with other servers.

- There are three rules for clients (one client, zero clients, multiple clients):

$$\frac{Q \vdash \Delta, y : A}{\overline{x}(y).Q \vdash \Delta, x : ?A}\ [?] \qquad \frac{Q \vdash \Delta}{Q \vdash \Delta, x : ?A}\ [W] \qquad \frac{Q \vdash \Delta, x : ?A, x' : ?A}{Q\{x/x'\} \vdash \Delta, x : ?A}\ [C]$$

# Clients and Servers in CLL

- There is one rule for servers:

$$\frac{P \vdash ?\Gamma, y : A}{!x(y).P \vdash ?\Gamma, x : !A}\ [!]$$

  $?\Gamma$ ensures: A process may only provide a server if it is implemented by communicating only with other servers.

- There are three rules for clients (one client, zero clients, multiple clients):

$$\frac{Q \vdash \Delta, y : A}{\overline{x}(y).Q \vdash \Delta, x : ?A}\ [?] \qquad \frac{Q \vdash \Delta}{Q \vdash \Delta, x : ?A}\ [W] \qquad \frac{Q \vdash \Delta, x : ?A, x' : ?A}{Q\{x/x'\} \vdash \Delta, x : ?A}\ [C]$$

- Accordingly, we have three reductions with a server $!x(y).P$ implementing $!A$:

$$(\boldsymbol{\nu} x)(!x(y).P \mid \overline{x}(y).Q) \longrightarrow (\boldsymbol{\nu} y)(P \mid Q)$$
$$(\boldsymbol{\nu} x)(!x(y).P \mid Q) \longrightarrow Q$$
$$(\boldsymbol{\nu} x)(!x(y).P \mid Q\{x/x'\}) \longrightarrow (\boldsymbol{\nu} x)(!x(y).P \mid (\boldsymbol{\nu} x')(!x'(y).P \mid Q))$$

# Taking Stock

- 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
- Deadlock freedom (DF)
- Asynchronous (classical) processes and DF.
- Clients and servers in DILL (and CLL).