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 29, 2026)

Asynchronous Communication

Outline

Up to Here

- Concurrent processes, session types, Linear Logic (LL).
- Concurrent interpretation of LL: statics and dynamics.
- Left and right rules give a **rely/guarantee** reading, per connective:
  - The right rule says how to **offer** a behavior specified by the connective;
  - The left rule says how to **use** such a behavior.
- Cut reductions and process synchronizations.
- The correctness properties ensured by the logic-based type system, in particular deadlock freedom (DF).
- The computational interpretation of proof transformations (reduction, but also structural congruence and commuting conversions).

Deadlock Freedom

- The type system derived from LL avoids deadlocks!

Deadlock Freedom

- The type system derived from LL avoids deadlocks!
- An example of a stuck process (different colors = different protocols):

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

A circular dependency:
the red input **blocks** the blue output; also,

Deadlock Freedom

- The type system derived from LL avoids deadlocks!
- An example of a stuck process (different colors = different protocols):

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

A circular dependency:
the red input **blocks** the blue output; also, the blue input **blocks** the red output.

Deadlock Freedom

- The type system derived from LL avoids deadlocks!
- An example of a stuck process (different colors = different protocols):

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

  A circular dependency:
  the red input **blocks** the blue output; also, the blue input **blocks** the red output.
- A similar process without this circular dependency:

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

  Still, Q cannot be typed: .

Deadlock Freedom

- The type system derived from LL avoids deadlocks!
- An example of a stuck process (different colors = different protocols):

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

  A circular dependency:
  the red input **blocks** the blue output; also, the blue input **blocks** the red output.

- A similar process without this circular dependency:

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

  Still, Q cannot be typed: sub-processes L and R share **two** channels (x and z).

Two Communication Mechanisms

- Our process model has **blocking actions** to induce sequential patterns specified by protocols. Deadlocks arise from issues in how actions block each other.

Two Communication Mechanisms

- Our process model has **blocking actions** to induce sequential patterns specified by protocols. Deadlocks arise from issues in how actions block each other.
- We have worked in a **synchronous** setting, where both outputs and inputs are blocking:

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

Before reduction, P and Q are blocked; after reduction, they are enabled.

Two Communication Mechanisms

- Our process model has **blocking actions** to induce sequential patterns specified by protocols. Deadlocks arise from issues in how actions block each other.
- We have worked in a **synchronous** setting, where both outputs and inputs are blocking:

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

  Before reduction, P and Q are blocked; after reduction, they are enabled.
- In an **asynchronous** setting, communication is simpler—only inputs are blocking:

$$\overline{x}\langle z\rangle.0 \mid x(y).Q \longrightarrow Q\{^z/_y\}$$

  The output does not block any actions (only 0), the actions in Q are enabled after reduction.

The Essentials of DF

We consider a simpler setting to investigate DF, in two steps:

1. We move to the concurrent interpretation of **classical** linear logic (CLL): the same expressive power as ILL, with less rules.[1]

[1] Philosophical considerations apply.

The Essentials of DF

We consider a simpler setting to investigate DF, in two steps:

1. We move to the concurrent interpretation of **classical** linear logic (CLL): the same expressive power as ILL, with less rules.[1]
2. We move to a process model with **asynchronous** communication, equipped with a continuation-passing mechanism to express sequential protocols.

[1] Philosophical considerations apply.

Outline

Classical Linear Logic / CP

Propositions and their Concurrent Interpretation (Wadler's Classical Processes):

$$
\begin{array}{rcll}
A, B & ::= & A \otimes B & \text{send} \\
& \mid & A \parr B & \text{receive} \\
& \mid & \oplus\{i : A\}_{i \in I} & \text{selection} \\
& \mid & \&\{i : A\}_{i \in I} & \text{branching} \\
& \mid & 1 & \text{closed protocol} \\
& \mid & \bot & \text{closed protocol}
\end{array}
$$

Classical Linear Logic

The **dual** of a proposition (session type) A, denoted $\overline{A}$:

$$\overline{A \otimes B} \triangleq \overline{A} \parr \overline{B} \qquad \overline{\oplus\{i : A_i\}_{i \in I}} \triangleq \&\{i : \overline{A_i}\}_{i \in I}$$

$$\overline{A \parr B} \triangleq \overline{A} \otimes \overline{B} \qquad \overline{\&\{i : A_i\}_{i \in I}} \triangleq \oplus\{i : \overline{A_i}\}_{i \in I}$$

$$\overline{1} \triangleq \bot \qquad \overline{\bot} \triangleq 1$$

The Process Language (Synchronous)

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

The Process Language (Synchronous)

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

The Process Language (Synchronous)

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

A useful notation: We write $\overline{x}(y).(P \mid Q)$ instead of $(\boldsymbol{\nu} y)\overline{x}\langle y\rangle.(P \mid Q)$.

Reduction

[red-send-recv]

$$\frac{}{(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \longrightarrow (\boldsymbol{\nu} uw)(P \mid (\boldsymbol{\nu} xy)(Q \mid R))}$$

Reduction

[red-send-recv]

$$\frac{}{(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \longrightarrow (\boldsymbol{\nu} uw)(P \mid (\boldsymbol{\nu} xy)(Q \mid R))}$$

[red-sel-bra]

$$\frac{j \in I}{(\boldsymbol{\nu} xy)(x \triangleleft \mathtt{l}_j; P \mid y \triangleright \{\mathtt{l}_1 : Q_1, \ldots, \mathtt{l}_n : Q_n\}) \longrightarrow (\boldsymbol{\nu} xy)(P \mid Q_j)}$$

[red-fwd]

$$\frac{y \neq z}{(\boldsymbol{\nu} xy)([x \leftarrow z] \mid P) \longrightarrow P\{^z/_y\}}$$

Reduction

[red-send-recv]

$$(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \longrightarrow (\boldsymbol{\nu} uw)(P \mid (\boldsymbol{\nu} xy)(Q \mid R))$$

[red-sel-bra]

$$\frac{j \in I}{(\boldsymbol{\nu} xy)(x \triangleleft \mathtt{l}_j; P \mid y \triangleright \{\mathtt{l}_1 : Q_1, \ldots, \mathtt{l}_n : Q_n\}) \longrightarrow (\boldsymbol{\nu} xy)(P \mid Q_j)}$$

[red-fwd]

$$\frac{y \neq z}{(\boldsymbol{\nu} xy)([x \leftarrow z] \mid P) \longrightarrow P\{^z/_y\}}$$

[red-sc]

$$\frac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q' \equiv Q}{P \longrightarrow Q}$$

[red-res]

$$\frac{P \longrightarrow Q}{(\boldsymbol{\nu} xy)P \longrightarrow (\boldsymbol{\nu} xy)Q}$$

[red-par]

$$\frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}$$

Typing Judgments

Judgments are of the form

$$P \vdash \Gamma$$

where

- P is a process
- Γ records endpoint/protocol assignments of the form $x : A$.

Typing Judgments

Judgments are of the form

$$P \vdash \Gamma$$

where

- P is a process
- Γ records endpoint/protocol assignments of the form $x : A$.

The context Γ disallows

- *weakening* (all assignments must be used)
- *contraction* (assignments may not be duplicated).

but obeys *exchange* (assignments may be silently reordered).

The empty context is written $\varnothing$.

Typing Rules based on CLL

$$\frac{}{[x \leftarrow y] \vdash x : \overline{A}, y : A}\ \text{[typ-fwd]}$$

Typing Rules based on CLL

[typ-fwd]

$$\frac{}{[x \leftarrow y] \vdash x : \overline{A}, y : A}$$

[typ-cut]

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : \overline{A}}{(\boldsymbol{\nu} xy)(P \mid Q) \vdash \Gamma, \Delta}$$

Typing Rules based on CLL

[typ-fwd]

$$\frac{}{[x \leftarrow y] \vdash x : \overline{A}, y : A}$$

[typ-cut]

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : \overline{A}}{(\boldsymbol{\nu} xy)(P \mid Q) \vdash \Gamma, \Delta}$$

[typ-send]

$$\frac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{\overline{x}(y).(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}$$

Typing Rules based on CLL

[typ-fwd]

$$\frac{}{[x \leftarrow y] \vdash x : \overline{A}, y : A}$$

[typ-cut]

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : \overline{A}}{(\boldsymbol{\nu} xy)(P \mid Q) \vdash \Gamma, \Delta}$$

[typ-send]

$$\frac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{\overline{x}(y).(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}$$

[typ-recv]

$$\frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B}$$

Typing Rules based on CLL

[typ-fwd]

$$\frac{}{[x \leftarrow y] \vdash x : \overline{A}, y : A}$$

[typ-cut]

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : \overline{A}}{(\boldsymbol{\nu} xy)(P \mid Q) \vdash \Gamma, \Delta}$$

[typ-send]

$$\frac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{\overline{x}(y).(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}$$

[typ-recv]

$$\frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B}$$

[typ-sel]

$$\frac{P \vdash \Gamma, x : A_j \qquad j \in I}{x \triangleleft \mathtt{l}_j; P \vdash x : \oplus\{i : A_i\}_{i \in I}}$$

[typ-bra]

$$\frac{\forall i \in I : P_i \vdash \Gamma, x : A_i}{x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\} \vdash \Gamma, x : \&\{i : A_i\}_{i \in I}}$$

[typ-inact]

$$\frac{}{0 \vdash \varnothing}$$

Typing Rules based on CLL

[typ-fwd]

$$\frac{}{[x \leftarrow y] \vdash x : \overline{A}, y : A}$$

[typ-cut]

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : \overline{A}}{(\boldsymbol{\nu} xy)(P \mid Q) \vdash \Gamma, \Delta}$$

[typ-send]

$$\frac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{\overline{x}(y).(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}$$

[typ-recv]

$$\frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B}$$

[typ-sel]

$$\frac{P \vdash \Gamma, x : A_j \qquad j \in I}{x \triangleleft \mathtt{l}_j; P \vdash x : \oplus\{i : A_i\}_{i \in I}}$$

[typ-bra]

$$\frac{\forall i \in I : P_i \vdash \Gamma, x : A_i}{x \triangleright \{\mathtt{l}_1 : P_1, \ldots, \mathtt{l}_n : P_n\} \vdash \Gamma, x : \&\{i : A_i\}_{i \in I}}$$

[typ-inact]

$$\frac{}{0 \vdash \varnothing}$$

[typ-one]

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

[typ-bot]

$$\frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \bot}$$

Example: Principal Cut Reduction for $\otimes/\parr$

$$\dfrac{\dfrac{P \vdash \Gamma_1, u : A \qquad Q \vdash \Gamma_2, x : B}{\overline{x}(u).(P \mid Q) \vdash \Gamma_1, \Gamma_2, x : A \otimes B} \qquad \dfrac{R \vdash \Delta, w : \overline{A}, y : \overline{B}}{y(w).R \vdash \Delta, y : \overline{A} \parr \overline{B}}}{(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \vdash \Gamma_1, \Gamma_2, \Delta}$$

$$\longrightarrow$$

Example: Principal Cut Reduction for $\otimes/\parr$

$$\dfrac{\dfrac{P \vdash \Gamma_1, u : A \qquad Q \vdash \Gamma_2, x : B}{\overline{x}(u).(P \mid Q) \vdash \Gamma_1, \Gamma_2, x : A \otimes B} \qquad \dfrac{R \vdash \Delta, w : \overline{A}, y : \overline{B}}{y(w).R \vdash \Delta, y : \overline{A} \parr \overline{B}}}{(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \vdash \Gamma_1, \Gamma_2, \Delta}$$

$$\longrightarrow$$

$$\dfrac{Q \vdash \Gamma_2, x : B \qquad R \vdash \Delta, w : \overline{A}, y : \overline{B}}{\qquad\qquad}$$

Example: Principal Cut Reduction for $\otimes/\parr$

$$\dfrac{\dfrac{P \vdash \Gamma_1, u : A \qquad Q \vdash \Gamma_2, x : B}{\overline{x}(u).(P \mid Q) \vdash \Gamma_1, \Gamma_2, x : A \otimes B} \qquad \dfrac{R \vdash \Delta, w : \overline{A}, y : \overline{B}}{y(w).R \vdash \Delta, y : \overline{A} \parr \overline{B}}}{(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \vdash \Gamma_1, \Gamma_2, \Delta}$$

$$\longrightarrow$$

$$\dfrac{Q \vdash \Gamma_2, x : B \qquad R \vdash \Delta, w : \overline{A}, y : \overline{B}}{(\boldsymbol{\nu} xy)(Q \mid R) \vdash \Gamma_2, \Delta, w : \overline{A}}$$

Example: Principal Cut Reduction for $\otimes/\parr$

$$\dfrac{\dfrac{P \vdash \Gamma_1, u : A \qquad Q \vdash \Gamma_2, x : B}{\overline{x}(u).(P \mid Q) \vdash \Gamma_1, \Gamma_2, x : A \otimes B} \qquad \dfrac{R \vdash \Delta, w : \overline{A}, y : \overline{B}}{y(w).R \vdash \Delta, y : \overline{A} \parr \overline{B}}}{(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \vdash \Gamma_1, \Gamma_2, \Delta}$$

$$\longrightarrow$$

$$\dfrac{P \vdash \Gamma_1, u : A \qquad \dfrac{Q \vdash \Gamma_2, x : B \qquad R \vdash \Delta, w : \overline{A}, y : \overline{B}}{(\boldsymbol{\nu} xy)(Q \mid R) \vdash \Gamma_2, \Delta, w : \overline{A}}}{(\boldsymbol{\nu} uw)(P \mid (\boldsymbol{\nu} xy)(Q \mid R)) \vdash \Gamma_1, \Gamma_2, \Delta}$$

Example: Principal Cut Reduction for $\otimes/\parr$

$$\dfrac{\dfrac{P \vdash \Gamma_1, u : A \qquad Q \vdash \Gamma_2, x : B}{\overline{x}(u).(P \mid Q) \vdash \Gamma_1, \Gamma_2, x : A \otimes B} \qquad \dfrac{R \vdash \Delta, w : \overline{A}, y : \overline{B}}{y(w).R \vdash \Delta, y : \overline{A} \parr \overline{B}}}{(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \vdash \Gamma_1, \Gamma_2, \Delta}$$

$$\longrightarrow$$

$$\dfrac{P \vdash \Gamma_1, u : A \qquad \dfrac{Q \vdash \Gamma_2, x : B \qquad R \vdash \Delta, w : \overline{A}, y : \overline{B}}{(\boldsymbol{\nu} xy)(Q \mid R) \vdash \Gamma_2, \Delta, w : \overline{A}}}{(\boldsymbol{\nu} uw)(P \mid (\boldsymbol{\nu} xy)(Q \mid R)) \vdash \Gamma_1, \Gamma_2, \Delta}$$

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

$$(\boldsymbol{\nu} xy)(\overline{x}(u).(P \mid Q) \mid y(w).R) \longrightarrow (\boldsymbol{\nu} uw)(P \mid (\boldsymbol{\nu} xy)(Q \mid R))$$

Outline

Some Observations

- Asynchronous communication is the cleanest setting to study DF by typing

Some Observations

- Asynchronous communication is the cleanest setting to study DF by typing
- Asynchrony unblocks behaviors / session types constrain behaviors

Some Observations

- Asynchronous communication is the cleanest setting to study DF by typing
- Asynchrony unblocks behaviors / session types constrain behaviors
- Asynchronous sessions “in between” untyped synchrony and asynchrony

Some Observations

- Asynchronous communication is the cleanest setting to study DF by typing
- Asynchrony unblocks behaviors / session types constrain behaviors
- Asynchronous sessions “in between” untyped synchrony and asynchrony
  - actions from different sessions are independent
  - actions of the same session should respect their session type

Some Observations

- Asynchronous communication is the cleanest setting to study DF by typing
- Asynchrony unblocks behaviors / session types constrain behaviors
- Asynchronous sessions "in between" untyped synchrony and asynchrony
  - actions from different sessions are independent
  - actions of the same session should respect their session type

Next An overview of DF enforcement in three typed asynchronous π-calculi:

Some Observations

- Asynchronous communication is the cleanest setting to study DF by typing
- Asynchrony unblocks behaviors / session types constrain behaviors
- Asynchronous sessions "in between" untyped synchrony and asynchrony
  - actions from different sessions are independent
  - actions of the same session should respect their session type

Next An overview of DF enforcement in three typed asynchronous π-calculi:

1. AP: processes follow protocols but freely deadlock (**no DF enforcement**)

Some Observations

- Asynchronous communication is the cleanest setting to study DF by typing
- Asynchrony unblocks behaviors / session types constrain behaviors
- Asynchronous sessions "in between" untyped synchrony and asynchrony
  - actions from different sessions are independent
  - actions of the same session should respect their session type

Next An overview of DF enforcement in three typed asynchronous π-calculi:

1. AP: processes follow protocols but freely deadlock (**no DF enforcement**)
2. ACP: DF by typing for tree-like process topologies (**basic enforcement**)

Some Observations

- Asynchronous communication is the cleanest setting to study DF by typing
- Asynchrony unblocks behaviors / session types constrain behaviors
- Asynchronous sessions "in between" untyped synchrony and asynchrony
  - actions from different sessions are independent
  - actions of the same session should respect their session type

Next An overview of DF enforcement in three typed asynchronous π-calculi:

1. AP: processes follow protocols but freely deadlock (**no DF enforcement**)
2. ACP: DF by typing for tree-like process topologies (**basic enforcement**)
3. APCP: DF also for circular process topologies (**advanced enforcement**)

An Asynchronous Process Language

$$
\begin{array}{rcll}
P, Q & ::= & x[a, b] & \text{send: both } a \text{ and } b \text{ are free} \\
& \mid & x(y, z); P & \text{receive} \\
& \mid & P \mid Q & \text{parallel} \\
& \mid & (\nu xy)P & \text{restriction} \\
& \mid & 0 & \text{inaction} \\
& \mid & [x \leftrightarrow y] & \text{forwarder} \\
& \mid & \cdots &
\end{array}
$$

An Asynchronous Process Language

$$
\begin{array}{rcll}
P, Q & ::= & x[a, b] & \text{send: both } a \text{ and } b \text{ are free} \\
& \mid & x(y, z); P & \text{receive} \\
& \mid & P \mid Q & \text{parallel} \\
& \mid & (\nu xy)P & \text{restriction} \\
& \mid & 0 & \text{inaction} \\
& \mid & [x \leftrightarrow y] & \text{forwarder} \\
& \mid & \ldots &
\end{array}
$$

Conventions:

- $a, b, c, \ldots, x, y, z, \ldots$ denote **names** (or **endpoints**)
- Early letters of the alphabet denote **objects** of output-like constructs.
- $\tilde{x}, \tilde{y}, \tilde{z}, \ldots$ denote finite sequences of names.

Reduction

[red-send-recv]

$$\frac{}{(\nu xy)(x[a, b] \mid y(a', b'); Q) \longrightarrow Q\{a/a', b/b'\}}$$

Reduction

[red-send-recv]

$$\frac{}{(\nu xy)(x[a, b] \mid y(a', b'); Q) \longrightarrow Q\{a/a', b/b'\}}$$

[red-sel-bra]

$$\frac{j \in I}{(\nu xy)(x[b] \triangleleft j \mid y(b') \triangleright \{i : Q_i\}_{i \in I}) \longrightarrow Q_j\{b/b'\}}$$

[red-fwd]

$$\frac{y \neq z}{(\nu xy)([x \leftrightarrow z] \mid P) \longrightarrow P\{z/y\}}$$

Reduction

[red-send-recv]

$$(\nu xy)(x[a, b] \mid y(a', b'); Q) \longrightarrow Q\{a/a', b/b'\}$$

[red-sel-bra]

$$\frac{j \in I}{(\nu xy)(x[b] \triangleleft j \mid y(b') \triangleright \{i : Q_i\}_{i \in I}) \longrightarrow Q_j\{b/b'\}}$$

[red-fwd]

$$\frac{y \neq z}{(\nu xy)([x \leftrightarrow z] \mid P) \longrightarrow P\{z/y\}}$$

[red-sc]

$$\frac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q' \equiv Q}{P \longrightarrow Q}$$

[red-res]

$$\frac{P \longrightarrow Q}{(\nu xy)P \longrightarrow (\nu xy)Q}$$

[red-par]

$$\frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}$$

Derived Constructs / Syntactic Sugar

$$\overline{x}[y] \cdot P \triangleq (\nu ya)(\nu zb)(x[a, b] \mid P\{z/x\})$$
$$x(y); P \triangleq x(y, z); P\{z/x\}$$
$$\overline{x} \triangleleft \ell \cdot P \triangleq (\nu zb)(x[b] \triangleleft \ell \mid P\{z/x\})$$
$$x \triangleright \{i : P_i\}_{i \in I} \triangleq x(z) \triangleright \{i : P_i\{z/x\}\}_{i \in I}$$

Note: We use '$\cdot$' to signify that output-like operations are non blocking.

Continuations Induce Session Protocols

$$
\begin{aligned}
P \triangleq (\nu zu)\Big((\nu xy)\Big(&(\nu ax')(x[v_1, a] \mid x'[v_2, b]) \\
&\mid (\nu cz')(z[v_3, c] \mid y(w_1, y'); y'(w_2, y''); Q)\Big) \\
&\mid u(w_3, u'); R\Big)
\end{aligned}
$$

Continuations Induce Session Protocols

$$P \triangleq (\nu zu)\Big((\nu xy)\Big((\nu ax')(x[v_1, a] \mid x'[v_2, b]) \\ \mid (\nu cz')(z[v_3, c] \mid y(w_1, y'); y'(w_2, y''); Q)\Big) \\ \mid u(w_3, u'); R\Big)$$

Using a **sugared syntax**:

$$P = (\nu zu)((\nu xy)(\overline{x}[v_1] \cdot \overline{x}[v_2] \cdot 0 \mid \overline{z}[v_3] \cdot y(w_1); y(w_2); Q') \mid u(w_3); R')$$

Continuations Induce Session Protocols

$$P \triangleq (\nu zu)\Big((\nu xy)\Big((\nu ax')(x[v_1, a] \mid x'[v_2, b]) \\ \mid (\nu cz')(z[v_3, c] \mid y(w_1, y'); y'(w_2, y''); Q)\Big) \\ \mid u(w_3, u'); R\Big)$$

Using a **sugared syntax**:

$$P = (\nu zu)((\nu xy)(\overline{x}[v_1] \cdot \overline{x}[v_2] \cdot 0 \mid \overline{z}[v_3] \cdot y(w_1); y(w_2); Q') \mid u(w_3); R')$$

We have independent reductions:

$$P \longrightarrow (\nu zu)((\nu xy)(\overline{x}[v_2] \cdot 0 \mid \overline{z}[v_3] \cdot y(w_2); Q'\{v_1/w_1\}) \mid u(w_3); R')$$
$$P \longrightarrow (\nu xy)(\overline{x}[v_1] \cdot \overline{x}[v_2] \cdot 0 \mid y(w_1); y(w_2); Q') \mid R'\{v_3/w_3\}$$

Note: There is no reduction involving the send on x': because x' is connected to the continuation of the send on x, it is thus not (yet) paired with a dual receive.

A Deadlocked Process, Now Asynchronous

The **cyclic dependency** between the two threads, now in the asynchronous case:

$$(\nu xy)(\nu uw)(x(v, x'); u[a, b] \mid w(z, w'); y[c, d])$$

Process is stuck:

- The input on x is waiting for the right output on y...
- ...the output on y is blocked by a receive (on w) waiting for the left output on u, which is blocked by the input on x.

Session Types

Types specify protocols. They (mostly) correspond to Linear Logic propositions:

$$
\begin{array}{rcll}
A, B & ::= & A \otimes B & \text{send} \\
& \mid & A \parr B & \text{receive} \\
& \mid & \bullet & \text{closed protocol} \\
& \mid & \cdots &
\end{array}
$$

Session Types

Types specify protocols. They (mostly) correspond to Linear Logic propositions:

$$
\begin{array}{rcll}
A, B & ::= & A \otimes B & \text{send} \\
& \mid & A \parr B & \text{receive} \\
& \mid & \bullet & \text{closed protocol} \\
& \mid & \ldots &
\end{array}
$$

The **dual** of session type A, denoted $\overline{A}$:

$$
\begin{aligned}
\overline{A \otimes B} &\triangleq \overline{A} \parr \overline{B} \\
\overline{A \parr B} &\triangleq \overline{A} \otimes \overline{B} \\
\overline{\bullet} &\triangleq \bullet
\end{aligned}
$$

Typing Judgments

Judgments are of the form

$$P \vdash \Gamma$$

where

- P is a process
- Γ records endpoint/protocol assignments of the form $x : A$.

Typing Judgments

Judgments are of the form

$$P \vdash \Gamma$$

where

- P is a process
- Γ records endpoint/protocol assignments of the form $x : A$.

The context Γ disallows

- *weakening* (all assignments must be used, except names typed with $\bullet$)
- *contraction* (assignments may not be duplicated).

but obeys *exchange* (assignments may be silently reordered).

The empty context is written $\varnothing$.

AP: Selected Typing Rules

[typ-send]

$$\frac{}{x[a, b] \vdash x : A \otimes B, a : \overline{A}, b : \overline{B}}$$

[typ-recv]

$$\frac{P \vdash \Gamma, y : A, z : B}{x(y, z); P \vdash \Gamma, x : A \parr B}$$

AP: Selected Typing Rules

[typ-send]

$$\frac{}{x[a, b] \vdash x : A \otimes B, a : \overline{A}, b : \overline{B}}$$

[typ-recv]

$$\frac{P \vdash \Gamma, y : A, z : B}{x(y, z); P \vdash \Gamma, x : A \parr B}$$

[typ-par]

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta}$$

[typ-res]

$$\frac{P \vdash \Gamma, x : A, y : \overline{A}}{(\nu xy)P \vdash \Gamma}$$

AP: Selected Typing Rules

[typ-send]

$$\frac{}{x[a, b] \vdash x : A \otimes B, a : \overline{A}, b : \overline{B}}$$

[typ-recv]

$$\frac{P \vdash \Gamma, y : A, z : B}{x(y, z); P \vdash \Gamma, x : A \parr B}$$

[typ-par]

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta}$$

[typ-res]

$$\frac{P \vdash \Gamma, x : A, y : \overline{A}}{(\nu xy)P \vdash \Gamma}$$

[typ-fwd]

$$\frac{}{[x \leftrightarrow y] \vdash x : \overline{A}, y : A}$$

[typ-end]

$$\frac{P \vdash \Gamma}{P \vdash \Gamma, x : \bullet}$$

[typ-inact]

$$\frac{}{0 \vdash \varnothing}$$

Properties of AP

Well-typed processes respect their session protocols under reduction:

Theorem (Type Preservation for AP)

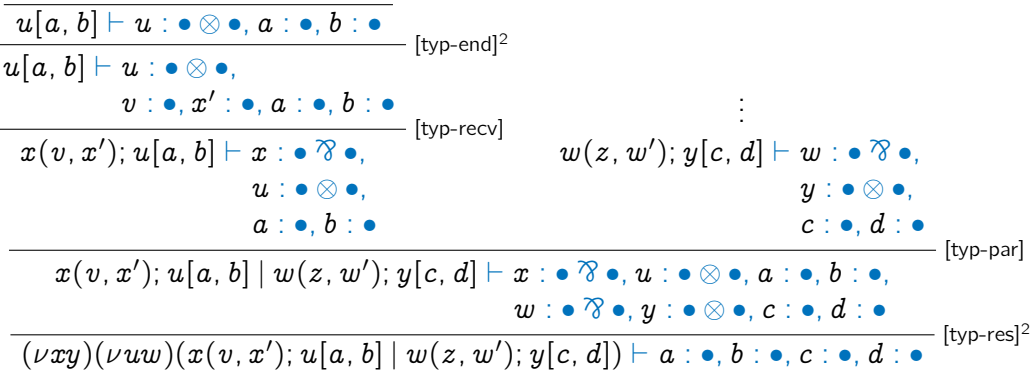
Given $P \vdash \Gamma$ *and* Q *such that* $P \equiv Q$ *or* $P \longrightarrow Q$, *we have* $Q \vdash \Gamma$.

Deadlocks are Typable in AP

Recall the process

$$(\nu xy)(\nu uw)(x(v, x'); u[a, b] \mid w(z, w'); y[c, d])$$

It can be typed as follows:



Outline

ACP: Asynchronous CP

- An asynchronous variant of Wadler's Classical Processes (CP).

ACP: Asynchronous CP

- An asynchronous variant of Wadler's Classical Processes (CP).
- Obtained from AP by a minor yet crucial modification.
  - **Remove**:

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta}\ [\text{typ-par}] \qquad \frac{P \vdash \Gamma, x : A, y : \overline{A}}{(\nu xy)P \vdash \Gamma}\ [\text{typ-res}]$$

  + **Add**:

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : \overline{A}}{(\nu xy)(P \mid Q) \vdash \Gamma, \Delta}\ [\text{typ-cut}]$$

ACP: Asynchronous CP

- An asynchronous variant of Wadler's Classical Processes (CP).
- Obtained from AP by a minor yet crucial modification.
  – **Remove**:

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta}\ [\text{typ-par}] \qquad \frac{P \vdash \Gamma, x : A, y : \overline{A}}{(\nu xy)P \vdash \Gamma}\ [\text{typ-res}]$$

  + **Add**:

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : \overline{A}}{(\nu xy)(P \mid Q) \vdash \Gamma, \Delta}\ [\text{typ-cut}]$$

- ACP guarantees DF by ruling out all possible cyclic dependencies: sub-processes can connect along **exactly one pair of names**.

ACP: Asynchronous CP

- An asynchronous variant of Wadler's Classical Processes (CP).
- Obtained from AP by a minor yet crucial modification.
  – **Remove**:

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta}\ [\text{typ-par}] \qquad \frac{P \vdash \Gamma, x : A, y : \overline{A}}{(\nu xy)P \vdash \Gamma}\ [\text{typ-res}]$$

  + **Add**:

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : \overline{A}}{(\nu xy)(P \mid Q) \vdash \Gamma, \Delta}\ [\text{typ-cut}]$$

- ACP guarantees DF by ruling out all possible cyclic dependencies: sub-processes can connect along **exactly one pair of names**.
- Note: We write $\overset{c}{\vdash}$ instead of $\vdash$ to distinguish the two type systems.

The Deadlocked Process, Revisited

- The deadlocked process

$$(\nu xy)(\nu uw)(x(v, x'); u[a, b] \mid w(z, w'); y[c, d])$$

is not typable under $\overset{c}{\vdash}$:
its parallel sub-processes are connected on two pairs of names.

The Deadlocked Process, Revisited

- The deadlocked process

$$(\nu xy)(\nu uw)(x(v, x'); u[a, b] \mid w(z, w'); y[c, d])$$

  is not typable under $\overset{C}{\vdash}$:
  its parallel sub-processes are connected on two pairs of names.

- A well-typed variant, obtained by 'parallelizing' the right sub-process:

$$(\nu xy)\Big((\nu uw)(x(v, x'); u[a, b] \mid w(z, w'); 0) \mid y[c, d]\Big) \overset{C}{\vdash} a : \bullet, b : \bullet, c : \bullet, d : \bullet$$

Properties of ACP

As in AP, well-typed processes respect their protocols:

Theorem (Type Preservation for ACP)

Given $P \overset{\mathtt{C}}{\vdash} \Gamma$ *and* Q *such that* $P \overset{\mathtt{C}}{\equiv} Q$ *or* $P \overset{\mathtt{C}}{\longrightarrow} Q$, *we have* $Q \overset{\mathtt{C}}{\vdash} \Gamma$.

Thanks to the cut rule, we now also have:

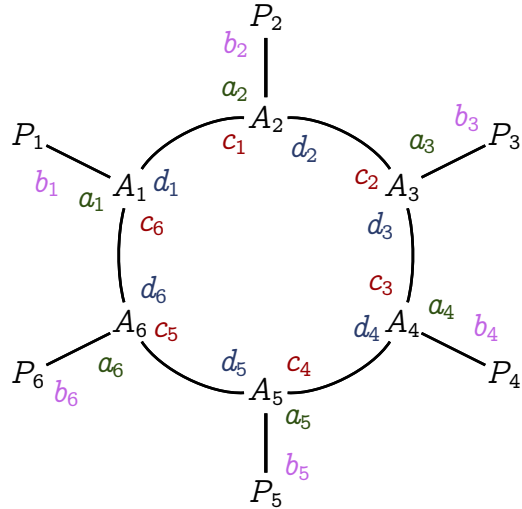
Theorem (Deadlock Freedom for ACP)

Given $P \overset{\mathtt{C}}{\vdash} \varnothing$, *if* $P \overset{\mathtt{C}}{\not\longrightarrow}$, *then* $P \overset{\mathtt{C}}{\equiv} 0$.

Not Typable in ACP: Milner's Cyclic Scheduler

Example of size 6:

$$(\nu\, c_1\, d_1)\cdots(\nu\, c_6\, d_6)\Big((\nu\, a_1\, b_1)(A_1 \mid P_1) \mid (\nu\, a_2\, b_2)(A_2 \mid P_2) \mid \vdots \mid (\nu\, a_6\, b_6)(A_6 \mid P_6)\Big)$$



APCP: Priority-Based DF Enforcement

Asynchronous **Priority-Based** Classical Processes:

- Back to **separate rules** for parallel composition and restriction (as in AP)
- Excluding cyclic dependencies using **priorities**

The Laws of Priorities

Key Idea

Typing ensures that each action has a corresponding priority (its 'urgency').
actions with **lower priority** must not be blocked by actions with **higher priority**.

The Laws of Priorities

Key Idea
Typing ensures that each action has a corresponding priority (its 'urgency').
actions with **lower priority** must not be blocked by actions with **higher priority**.

Write $\pi, \rho, \ldots$ to denote priorities (integers). Laws enforced by typing:

1. Outputs with priority π send messages with **larger priority** than π;
2. A action with priority π are prefixed by inputs with **smaller priority** than π;
3. Dual actions must have **equal priorities**.

APCP: Priority-Based DF Enforcement

- Types are as before, now annotated with priorities:

$$A, B ::= A \otimes^{\pi} B \mid A \parr^{\pi} B \mid \bullet \mid \cdots$$

APCP: Priority-Based DF Enforcement

- Types are as before, now annotated with priorities:

$$A, B ::= A \otimes^{\pi} B \mid A \parr^{\pi} B \mid \bullet \mid \cdots$$

- We write ω to denote the 'ultimate priority': it is greater than all other priorities and cannot be increased further.

APCP: Priority-Based DF Enforcement

- Types are as before, now annotated with priorities:

$$A, B ::= A \otimes^{\pi} B \mid A \parr^{\pi} B \mid \bullet \mid \cdots$$

- We write ω to denote the 'ultimate priority': it is greater than all other priorities and cannot be increased further.
- For session type A, $pr(A)$ denotes its *priority*:

$$pr(A \otimes^{\pi} B) \triangleq pr(A \parr^{\pi} B) \triangleq \pi$$
$$pr(\bullet) \triangleq \omega$$

APCP: Priority-Based DF Enforcement

- Types are as before, now annotated with priorities:

$$A, B ::= A \otimes^{\pi} B \mid A \parr^{\pi} B \mid \bullet \mid \cdots$$

- We write ω to denote the 'ultimate priority': it is greater than all other priorities and cannot be increased further.
- For session type A, $pr(A)$ denotes its *priority*:

$$pr(A \otimes^{\pi} B) \triangleq pr(A \parr^{\pi} B) \triangleq \pi$$
$$pr(\bullet) \triangleq \omega$$

- Duality: $A = \overline{B}$ iff (i) actions in A and B are complementary (as before) and (ii) their priorities coincide.

Typing rules of APCP: AP with Priorities

[typ-send]

$$\frac{\pi < pr(A), pr(B)}{x[y,z] \vdash^{\mathrm{P}} x : A \otimes^{\pi} B, y : \overline{A}, z : \overline{B}}$$

[typ-recv]

$$\frac{P \vdash^{\mathrm{P}} \Gamma, y : A, z : B \qquad \pi < pr(\Gamma)}{x(y,z); P \vdash^{\mathrm{P}} \Gamma, x : A \parr^{\pi} B}$$

(Other rules similar or unchanged, with the extended duality)

Properties of APCP

Theorem (Type Preservation for APCP)

Given $P \overset{\mathrm{P}}{\vdash} \Gamma$ *and* Q *such that* $P \equiv Q$ *or* $P \longrightarrow Q$, *we have* $Q \overset{\mathrm{P}}{\vdash} \Gamma$.

Theorem (Deadlock Freedom for APCP)

Given $P \overset{\mathrm{P}}{\vdash} \varnothing$, *if* $P \not\longrightarrow$, *then* $P \equiv 0$.

Examples

- Recall the deadlocked process; it is rejected under $\overset{\mathrm{P}}{\vdash}$:

$$(\nu xy)(\nu uw)(x(v, x'); u[a, b] \mid w(z, w'); y[c, d])$$

Suppose that actions on x and y have priority π, and that actions on u and w have priority ρ. Using Rule [typ-recv] we require $\pi < \rho$ and $\rho < \pi$.

Examples

- Recall the deadlocked process; it is rejected under $\overset{\mathrm{P}}{\vdash}$:

$$(\nu xy)(\nu uw)(x(v, x'); u[a, b] \mid w(z, w'); y[c, d])$$

  Suppose that actions on x and y have priority π, and that actions on u and w have priority ρ. Using Rule [typ-recv] we require $\pi < \rho$ and $\rho < \pi$.

- Recall the non-deadlocked variant:

$$(\nu xy)\Big((\nu uw)(x(v, x'); u[a, b] \mid w(z, w'); 0) \mid y[c, d]\Big)$$

  In this case, $\overset{\mathrm{P}}{\vdash}$ only requires $\pi < \rho$ but not $\rho < \pi$, so no cyclic dependency is detected—the process is considered well typed.

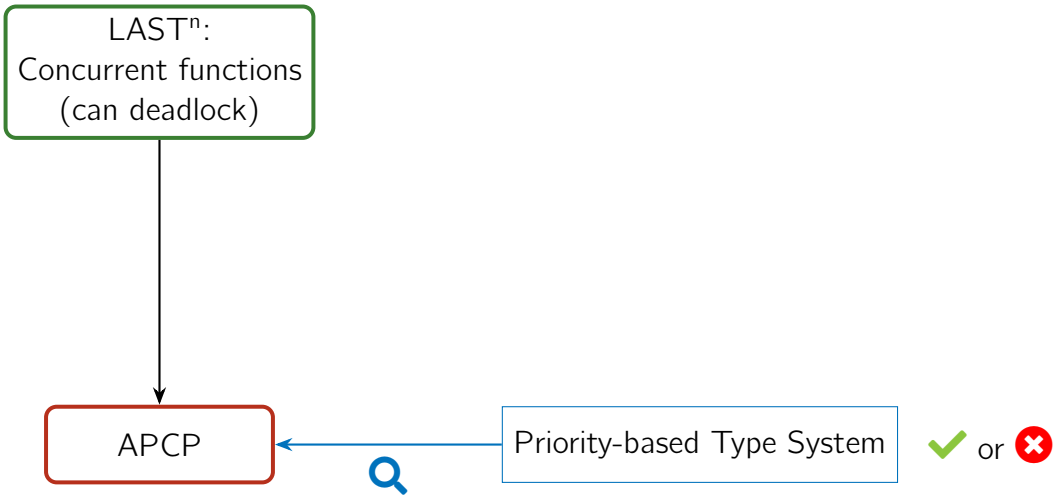
Application: DF in Concurrent Functional Sessions

LAST^{n}:
Concurrent functions
(can deadlock)

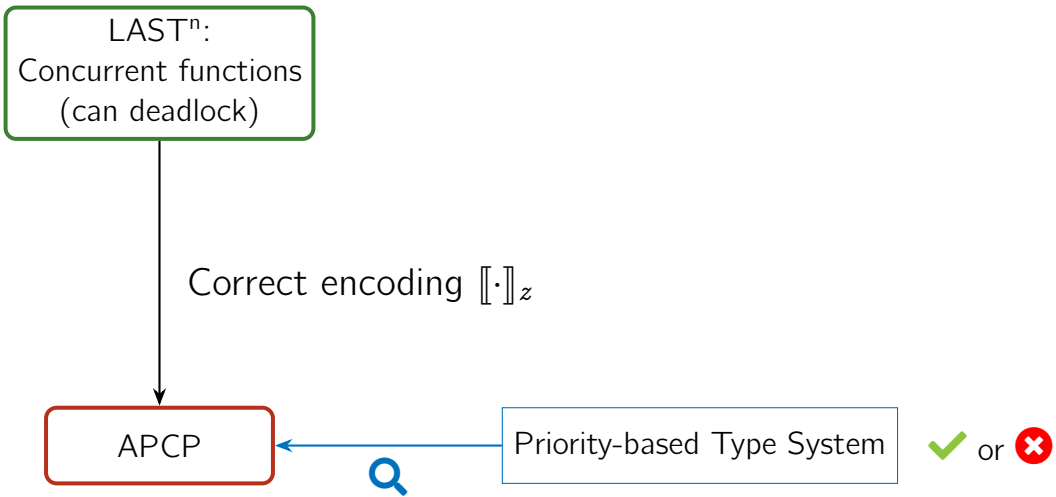
Application: DF in Concurrent Functional Sessions

LAST^n:
Concurrent functions
(can deadlock)

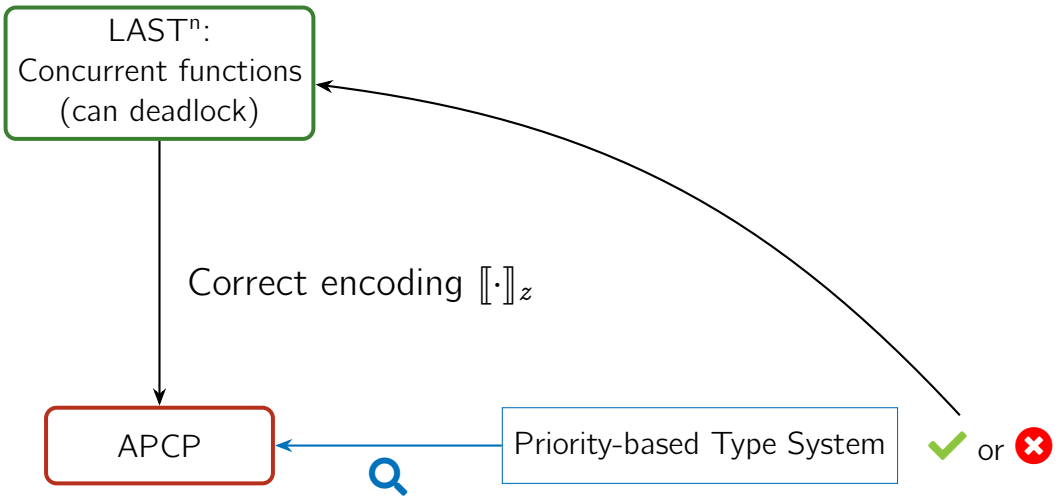
Application: DF in Concurrent Functional Sessions



Application: DF in Concurrent Functional Sessions



Application: DF in Concurrent Functional Sessions



Summing Up

The concurrent interpretation of Linear Logic, now in the classical presentation.

Three typed **asynchronous** π-calculi based on Linear Logic:

1. AP: processes correctly follow protocols but may deadlock
2. ACP: AP with logic-based composition, DF for tree-like topologies
3. APCP: extension of AP with priorities, DF for circular topologies