A Bunch of Sessions: Session Types beyond Linear Logic

Jorge A. Pérez (University of Groningen, The Netherlands)

joint work with
Emanuele D'Osualdo (MPI-SWS), Dan Frumin (Aarhus), and Bas van den Heuvel (Groningen)

UNIFYING C•RRECTNESS FOR C•MMUNICATING S•FTWARE

June 19, 2023

Context

Logic has long provided a solid basis for the principled analysis of programs.
Very productive, especially for programs involving **concurrency**:

- **Separation Logics** (O'Hearn and Reynolds):
  Exploit local reasoning to support useful forms of safe interference and sharing in heap-manipulating programs.
- **Linear Logic** (Girard):
  Flexible foundation for ensuring that message-passing processes correctly implement protocols, avoiding races and deadlocks.

Context

Logic has long provided a solid basis for the principled analysis of programs.
Very productive, especially for programs involving **concurrency**:

Separation Logics (O'Hearn and Reynolds):
Exploit local reasoning to support useful forms of safe interference and sharing in heap-manipulating programs.
Linear Logic (Girard):
Flexible foundation for ensuring that message-passing processes correctly implement protocols, avoiding races and deadlocks.

Propositions-as-Sessions [Caires & Pfenning; Wadler]

- ✓ Firm justification for seminal work on session types
- ✓ Reference framework for expressiveness
- ✓ Canonical platform for extensions (e.g., sharing)

This Talk: Session Types beyond Linear Logic

Key ideas:

- **Bunched Implications** (**BI**, O'Hearn and Pym): the basis of (concurrent) separation logics
- Shift from **consuming resources** (LL) to **accessing them** (BI).
- LL and BI are *incomparable*. Alternative angle to non-linear (shared) resources (e.g. client/server channels)

This Talk: Session Types beyond Linear Logic

Main result:

A **concurrent interpretation** of BI in the style of *Propositions-as-Sessions*.

This Talk: Session Types beyond Linear Logic

Main result:

A **concurrent interpretation** of BI in the style of *Propositions-as-Sessions*.

Key contributions (OOPSLA’22):

- ✓ The process language πBI, and its *spawn construct*
- ✓ A new treatment of structural rules that justifies spawn
- ✓ Type preservation, deadlock-freedom, weak normalization.
- ✓ Denotational semantics and observational equivalences
- ✓ An encoding of O’Hearn and Pym’s $\alpha\lambda$-calculus into πBI

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic:

Logic	$\leftrightarrow$	**Programming Languages**
Propositions	$\leftrightarrow$	Types
Proofs	$\leftrightarrow$	Programs
Proof simplification	$\leftrightarrow$	Computation steps
Cut-elimination	$\leftrightarrow$	Normalization properties

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Sequential interpretations

Intuitionistic Logic $\leftrightarrow$

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Sequential interpretations

Intuitionistic Logic $\leftrightarrow$ Simply-typed λ-calculus

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \boldsymbol{\lambda} x{:}\, t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}$$

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Sequential interpretations

Intuitionistic Logic $\leftrightarrow$ Simply-typed λ-calculus

Linear Logic (LL) $\leftrightarrow$ Linear λ-calculus

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Sequential interpretations

Intuitionistic Logic	$\leftrightarrow$	Simply-typed λ-calculus
Linear Logic (LL)	$\leftrightarrow$	Linear λ-calculus
Bunched Impl. (BI)	$\leftrightarrow$	$\alpha\lambda$-calculus

Concurrent interpretations

Linear Logic (LL)	$\leftrightarrow$	Session-typed π-calculus

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Concurrent interpretations

Linear Logic (LL)	$\leftrightarrow$	Session-typed π-calculus
propositions	$\leftrightarrow$	session types (protocols)
proofs	$\leftrightarrow$	processes
cut elimination	$\leftrightarrow$	communication

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Concurrent interpretations

Linear Logic (LL) $\leftrightarrow$ Session-typed π-calculus

Sample rules:

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

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

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Concurrent interpretations

Linear Logic (LL) $\leftrightarrow$ Session-typed π-calculus

Sample rules:

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

$$\frac{\Delta \,, y : A \,, x : B \vdash P :: z : C}{\Delta \,, x : A \otimes B \vdash x(y).P :: z : C} \qquad \frac{\Delta_1 \vdash P :: y : A \qquad \Delta_2 \vdash Q :: x : B}{\Delta_1 \,, \Delta_2 \vdash \overline{x}[y].(P \mid Q) :: x : A \otimes B}$$

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Concurrent interpretations

Linear Logic (LL)	$\leftrightarrow$	Session-typed π-calculus	
Bunched Impl. (BI)	$\leftrightarrow$	πBI	$\leftarrow$ THIS WORK

Logic of Bunched Implications

Bunched Implications (BI)

BI combines intuitionistic logic and (multiplicative) linear logic:

$$A, B \in \mathit{Prop} ::= \mathbf{1}_{\mathsf{a}} \mid A \vee B \mid A \wedge B \mid A \to B$$

Bunched Implications (BI)

BI combines intuitionistic logic and (multiplicative) linear logic:

$$
\begin{aligned}
A, B \in \mathit{Prop} ::= {} & \mathbf{1}_{\mathsf{a}} \mid A \vee B \mid A \wedge B \mid A \to B \\
& \mid \mathbf{1}_{\mathsf{m}} \mid A * B \mid A \mathrel{-\!*} B
\end{aligned}
$$

Intuitionistic logic

Bunched Implications (BI)

BI combines intuitionistic logic and (multiplicative) linear logic:

$$\begin{aligned} A, B \in \mathit{Prop} ::= {} & \mathbf{1}_{\mathsf{a}} \mid A \vee B \mid A \wedge B \mid A \rightarrow B \\ & \mid \mathbf{1}_{\mathsf{m}} \mid A * B \mid A \mathrel{-\!\!*} B \end{aligned}$$

Linear logic (fragment)

Bunched Implications (BI)

BI combines intuitionistic logic and (multiplicative) linear logic:

$$\begin{aligned} A, B \in \textit{Prop} ::= {} & \mathbf{1}_{\mathsf{a}} \mid A \vee B \mid A \wedge B \mid A \to B \\ & \mid \mathbf{1}_{\mathsf{m}} \mid A * B \mid A \mathrel{-\!\!*} B \end{aligned}$$

Propositions represent ownership of resources:

$A \wedge B$: both A and B are true of the resources we own

$A * B$: resources we own can be divided into A-resources and B-resources

Example: $\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2$

Bunched Implications (BI)

$$A \wedge (A \to B) \to B \qquad\qquad A * (A \mathrel{-\!\!*} B) \mathrel{-\!\!*} B$$

Bunched Implications (BI)

$$A \wedge (A \to B) \to B \qquad A * (A \mathrel{-\!\!*} B) \mathrel{-\!\!*} B$$

$$A \to A \wedge A \qquad A \not\mathrel{-\!\!*} A * A$$

$$A \wedge B \to A \qquad A * B \not\mathrel{-\!\!*} A$$

Bunched Implications (BI)

$$A \wedge (A \to B) \to B \qquad A * (A \mathrel{-\!\!*} B) \mathrel{-\!\!*} B$$

$$A \to A \wedge A \qquad A \not\mathrel{-\!\!*} A * A$$

$$A \wedge B \to A \qquad A * B \not\mathrel{-\!\!*} A$$

$$A \wedge (A \to B) \to A \wedge B \qquad A * (A \mathrel{-\!\!*} B) \not\mathrel{-\!\!*} A * B$$

A sequent calculus for BI

The sequent calculus for BI externalizes $\wedge$ and $*$ as different connectives, denoted '**;**' and '**,**'.

Crucially, only '**;**' admits weakening and contraction.

A sequent calculus for BI

The sequent calculus for BI externalizes $\wedge$ and $*$ as different connectives, denoted '**;**' and '**,**'.

Crucially, only '**;**' admits weakening and contraction.

As a result, contexts in the sequents are not lists/multisets, but trees with nodes '**;**' and '**,**' aka **bunches**.

A sequent calculus for BI

$\boxed{\text{Sequent: } \Gamma \vdash \phi}$

$$\frac{\Gamma; A; B \vdash C}{\Gamma; A \wedge B \vdash C} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1; \Gamma_2 \vdash A \wedge B}$$

A sequent calculus for BI

Left and right rules

$$\frac{\Gamma; A; B \vdash C}{\Gamma; A \wedge B \vdash C} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1; \Gamma_2 \vdash A \wedge B}$$

$$\frac{\Gamma; \Gamma \vdash C}{\Gamma \vdash C} \qquad \frac{\Gamma \vdash C}{\Gamma; \Gamma' \vdash C}$$

A sequent calculus for BI

Structural rules

$$\frac{\Gamma; A \,\boldsymbol{;}\, B \vdash C}{\Gamma; A \wedge B \vdash C} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1 \,\boldsymbol{;}\, \Gamma_2 \vdash A \wedge B}$$

$$\frac{\Gamma \,\boldsymbol{;}\, \Gamma \vdash C}{\Gamma \vdash C} \qquad \frac{\Gamma \vdash C}{\Gamma \,\boldsymbol{;}\, \Gamma' \vdash C}$$

A sequent calculus for BI

$$\frac{\Gamma; A \,,\, B \vdash C}{\Gamma; A * B \vdash C} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1 \,,\, \Gamma_2 \vdash A * B}$$

$$\frac{\Gamma; A \,;\, B \vdash C}{\Gamma; A \wedge B \vdash C} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1 \,;\, \Gamma_2 \vdash A \wedge B}$$

$$\frac{\Gamma \,;\, \Gamma \vdash C}{\Gamma \vdash C} \qquad \frac{\Gamma \vdash C}{\Gamma \,;\, \Gamma' \vdash C}$$

A sequent calculus for BI

$$\boxed{\Gamma ::= A \mid \Gamma \mathbin{;} \Gamma \mid \Gamma \mathbin{,} \Gamma \mid \ldots}$$

$$\frac{\Delta(A \mathbin{,} B) \vdash C}{\Delta(A * B) \vdash C} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1 \mathbin{,} \Gamma_2 \vdash A * B}$$

$$\frac{\Delta(A \mathbin{;} B) \vdash C}{\Delta(A \wedge B) \vdash C} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1 \mathbin{;} \Gamma_2 \vdash A \wedge B}$$

$$\frac{\Delta(\Gamma \mathbin{;} \Gamma) \vdash C}{\Delta(\Gamma) \vdash C} \qquad \frac{\Delta(\Gamma) \vdash C}{\Delta(\Gamma \mathbin{;} \Gamma') \vdash C}$$

Left rules can be applied deep inside an arbitrary *bunched context*.

The Process Language

The πBI Process Language

$P, Q ::= \overline{x}[y].(P \mid Q)$	output	$\mid\ x(y).P$	input
$\mid\ \overline{x}\langle\rangle$	close	$\mid\ x().P$	wait
$\mid\ x \triangleleft \mathsf{inl}.P$	left select	$\mid\ x \triangleright \mathsf{case}(P, Q)$	branch
$\mid\ x \triangleleft \mathsf{inr}.P$	right select	$\mid\ [x \leftarrow y]$	forwarder
$\mid\ (\boldsymbol{\nu} x).(P \mid Q)$	composition		

The πBI Process Language

$P, Q ::= \overline{x}[y].(P \mid Q)$	output	$\mid\ x(y).P$	input
$\mid\ \overline{x}\langle\rangle$	close	$\mid\ x().P$	wait
$\mid\ x \triangleleft \mathsf{inl}.P$	left select	$\mid\ x \triangleright \mathsf{case}(P, Q)$	branch
$\mid\ x \triangleleft \mathsf{inr}.P$	right select	$\mid\ [x \leftarrow y]$	forwarder
$\mid\ (\boldsymbol{\nu} x).(P \mid Q)$	composition		
$\mid\ \boldsymbol{\rho}[\sigma].P$	spawn		

The Spawn Construct: Intuitions

A prefix specifying non-local replicated behavior, defined hand-in-hand with structural properties in BI.

The Spawn Construct: Intuitions

A prefix specifying non-local replicated behavior, defined hand-in-hand with structural properties in BI.

We write

$$\boldsymbol{\rho}[\sigma].P$$

where σ is a list of sessions that should be duplicated (contraction) or discarded (weakening) *by the context*.

[A rough analogy: horizontal scaling in cloud computing.]

The Spawn Construct: Intuitions

A prefix specifying non-local replicated behavior, defined hand-in-hand with structural properties in BI.

We write

$$\boldsymbol{\rho}[\sigma].P$$

where σ is a list of sessions that should be duplicated (contraction) or discarded (weakening) *by the context.*

[A rough analogy: horizontal scaling in cloud computing.]

Examples—different requests to an environment/context:

$\boldsymbol{\rho}[x \mapsto x_1, x_2].Q$	Duplicate x as x_1, x_2, then run Q
$\boldsymbol{\rho}[x \mapsto \emptyset].Q$	Drop x, then run Q
$\boldsymbol{\rho}[x \mapsto x_1, x_2, x_3\ ,\ y \mapsto \emptyset].Q$	Replicate x, drop y, run Q

Reduction Semantics

Most reduction rules follow *propositions-as-sessions*:

RED-COMM-R

$$(\boldsymbol{\nu} x).\big(x(y).Q \mid_x \overline{x}[y].(P_1 \mid P_2)\big) \longrightarrow (\boldsymbol{\nu} x).\big((\boldsymbol{\nu} y).(P_1 \mid_y Q) \mid_x P_2\big)$$

RED-UNIT-R

$$(\boldsymbol{\nu} x).\big(x().Q \mid_x \overline{x}\langle\rangle\big) \longrightarrow Q$$

RED-CASE

$$\frac{\ell \in \{\mathsf{inl}, \mathsf{inr}\}}{(\boldsymbol{\nu} x).\big(x \triangleleft \ell.P \mid x \triangleright \mathsf{case}(Q_{\mathsf{inl}}, Q_{\mathsf{inr}})\big) \longrightarrow (\boldsymbol{\nu} x).\big(P \mid Q_\ell\big)}$$

RED-FWD-R

$$(\boldsymbol{\nu} x).(P \mid_x [y \leftarrow x]) \longrightarrow P[y/x]$$

Reduction Semantics

Reduction for spawn is more interesting.
Consider process **duplication**:

$$(\boldsymbol{\nu} x).(P \mid_x \boldsymbol{\rho}[x \mapsto x_1, x_2].Q) \longrightarrow$$
$$(\boldsymbol{\nu} x_1).\big(P[x_1/x] \mid_{x_1} (\boldsymbol{\nu} x_2).(P[x_2/x] \mid_{x_2} Q)\big),$$

Note: Asfter reduction, Q gets two copies of P, properly adjusted.

Reduction Semantics

Reduction for spawn is more interesting.
Consider process **duplication**:

$$(\boldsymbol{\nu} x).(P \mid_x \boldsymbol{\rho}[x \mapsto x_1, x_2].Q) \longrightarrow$$
$$(\boldsymbol{\nu} x_1).\big(P[x_1/x] \mid_{x_1} (\boldsymbol{\nu} x_2).(P[x_2/x] \mid_{x_2} Q)\big),$$

Note: Asfter reduction, Q gets two copies of P, properly adjusted.

Now consider process **discarding**:

$$(\boldsymbol{\nu} x).(P \mid_x \boldsymbol{\rho}[x \mapsto \emptyset].Q) \longrightarrow Q$$

If P only communicates through x.

Reductions

Let's gradually get more precise:

RED-SPAWN*

$$\frac{\sigma(x) = \{x_1, x_2\}}{(\boldsymbol{\nu} x).\big(P \mid_x \boldsymbol{\rho}[\sigma].Q\big) \longrightarrow (\boldsymbol{\nu} x_1).\big(P^{(1)} \mid_{x_1} (\boldsymbol{\nu} x_2).(P^{(2)} \mid_{x_2} Q)\big)}$$

Reductions

Let's gradually get more precise:

RED-SPAWN*

$$\frac{\sigma(x) = \{x_1, x_2\}}{(\boldsymbol{\nu} x).\big(P \mid_x \boldsymbol{\rho}[\sigma].Q\big) \longrightarrow (\boldsymbol{\nu} x_1).\big(P^{(1)} \mid_{x_1} (\boldsymbol{\nu} x_2).(P^{(2)} \mid_{x_2} Q)\big)}$$

Note:

> This is *not enough*, because duplication on x has an effect also on other free names in P (different from x)

Reductions

Let's gradually get more precise:

RED-SPAWN*

$$\frac{\sigma(x) = \{x_1, x_2\}}{(\boldsymbol{\nu} x).\big(P \mid_x \boldsymbol{\rho}[\sigma].Q\big) \longrightarrow (\boldsymbol{\nu} x_1).\big(P^{(1)} \mid_{x_1} (\boldsymbol{\nu} x_2).(P^{(2)} \mid_{x_2} Q)\big)}$$

Note:

- This is *not enough*, because duplication on x has an effect also on other free names in P (different from x)
- The spawn prefix must survive reduction with the "rest" of σ, to signal the needed duplication in those names
- We need something else...

Reductions

Let's gradually get more precise:

RED-SPAWN*

$$\frac{\sigma(x) = \{x_1, x_2\} \quad \sigma' = \big((\sigma \setminus \{x\}) \cup [z \mapsto \{z_1, z_2\} \mid z \in \mathrm{fn}(P) \setminus \{x\}]\big)}{(\boldsymbol{\nu} x).\big(P \mid_x \boldsymbol{\rho}[\sigma].Q\big) \longrightarrow \boldsymbol{\rho}[\sigma'].(\boldsymbol{\nu} x_1).\big(P^{(1)} \mid_{x_1} (\boldsymbol{\nu} x_2).(P^{(2)} \mid_{x_2} Q)\big)}$$

Note:

- This is *not enough*, because duplication on x has an effect also on other free names in P (different from x)
- The spawn prefix must survive reduction with the "rest" of σ, to signal the needed duplication in those names
- We need something else...

Reduction Semantics

The general rule:

RED-SPAWN

$$\frac{\sigma(x) = \{x_1, \ldots, x_n\} \qquad \sigma' = \big((\sigma \setminus \{x\}) \cup [z \mapsto \{z_1, \ldots, z_n\} \mid z \in \mathrm{fn}(P) \setminus \{x\}]\big)}{(\boldsymbol{\nu} x).\big(P \mid_x \boldsymbol{\rho}[\sigma].Q\big) \longrightarrow \boldsymbol{\rho}[\sigma'].(\boldsymbol{\nu} x_1).\big(P^{(1)} \mid_{x_1} \ldots (\boldsymbol{\nu} x_n).(P^{(n)} \mid_{x_n} Q)\big)}$$

Reduction Semantics

The general rule:

RED-SPAWN

$$\frac{\sigma(x) = \{x_1, \dots, x_n\} \qquad \sigma' = \big((\sigma \setminus \{x\}) \cup [z \mapsto \{z_1, \dots, z_n\} \mid z \in \mathrm{fn}(P) \setminus \{x\}]\big)}{(\boldsymbol{\nu} x).\big(P \mid_x \boldsymbol{\rho}[\sigma].Q\big) \longrightarrow \boldsymbol{\rho}[\sigma'].(\boldsymbol{\nu} x_1).\big(P^{(1)} \mid_{x_1} \dots (\boldsymbol{\nu} x_n).(P^{(n)} \mid_{x_n} Q)\big)}$$

RED-SPAWN-R

$$\frac{x \notin \mathrm{dom}(\sigma)}{(\boldsymbol{\nu} x).\big(P \mid_x \boldsymbol{\rho}[\sigma].Q\big) \longrightarrow \boldsymbol{\rho}[\sigma].(\boldsymbol{\nu} x).(P \mid_x Q)\big)}$$

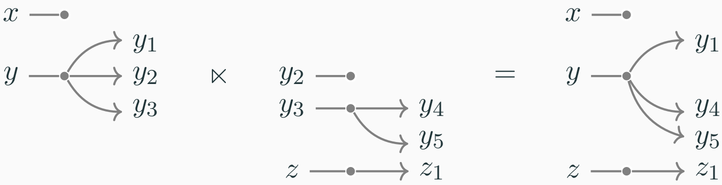
Reduction Semantics

We can "accumulate" spawn prefixes:

RED-SPAWN-MERGE

$$\boldsymbol{\rho}[\sigma_1].\boldsymbol{\rho}[\sigma_2].P \longrightarrow \boldsymbol{\rho}[\sigma_1 \ltimes \sigma_2].P$$

where $\ltimes$ is the **merge operator** that "connects" σ_1 and σ_2.
It operates as follows:



Congruences (Selection)

CONG-ASSOC-L

$$\frac{x \notin \mathrm{fn}(Q) \wedge y \notin \mathrm{fn}(P)}{(\boldsymbol{\nu} x).\big(P \mid_x (\boldsymbol{\nu} y).(Q \mid_y R)\big) \equiv (\boldsymbol{\nu} y).\big(Q \mid_y (\boldsymbol{\nu} x).(P \mid_x R)\big)}$$

CONGR-ASSOC-R

$$\frac{x \notin \mathrm{fn}(R) \wedge y \notin \mathrm{fn}(P)}{(\boldsymbol{\nu} x).\big(P \mid_x (\boldsymbol{\nu} y).(Q \mid_y R)\big) \equiv (\boldsymbol{\nu} y).\big((\boldsymbol{\nu} x).(P \mid_x Q) \mid_y R\big)}$$

CONGR-SPAWN-SWAP

$$\frac{\sigma_1 \# \sigma_2}{\boldsymbol{\rho}[\sigma_1].\boldsymbol{\rho}[\sigma_2].Q \equiv \boldsymbol{\rho}[\sigma_2].\boldsymbol{\rho}[\sigma_1].Q}$$

An Unusual Example

Consider the following πBI process:

$$P \triangleq z(a).z(y).\boldsymbol{\rho}[a \mapsto a_1, a_2].\overline{y}[a_1'].\big([a_1' \leftarrow a_1] \mid \overline{y}[a_2'].([a_2' \leftarrow a_2] \mid [z \leftarrow y])\big)$$

Intuitively:

- Process P receives a single session of type A over z through a *multiplicative* input (to be typed with $-\!\!*$).
- The session on y uses A *twice*; the two A-typed sessions share a *common origin*.
- P spawns two copies of A and use them to interact on y.

An Unusual Example

Consider the following πBI process:

$$P \triangleq z(a).z(y).\boldsymbol{\rho}[a \mapsto a_1, a_2].\overline{y}[a'_1].\big([a'_1 \leftarrow a_1] \mid \overline{y}[a'_2].([a'_2 \leftarrow a_2] \mid [z \leftarrow y])\big)$$

Intuitively:

- Process P receives a single session of type A over z through a *multiplicative* input (to be typed with $-\!\!*$).
- The session on y uses A *twice*; the two A-typed sessions share a *common origin*.
- P spawns two copies of A and use them to interact on y.

As we will see, the following judgment is derivable

$$\emptyset_{\mathsf{m}} \vdash P :: z : A -\!\!* (A \to A \to B) \to B$$

BI as a Type System for Sessions

Typing: Identity and Cut

We retain key judgmental principles from propositions-as-sessions.

Identity as forwarding:

$$\dfrac{\textsc{Fwd}}{y : A \vdash [x \leftarrow y] :: x : A}$$

Cut as composition:

$$\textsc{Cut}\ \dfrac{\Delta \vdash P :: x : A \qquad \Gamma(x : A) \vdash Q :: z : C}{\Gamma(\Delta) \vdash (\boldsymbol{\nu} x).(P \mid Q) :: z : C}$$

Typing: Multiplicative Connectives

Multiplicative conjunction and implication are interpreted as output and input...

$$\textsc{Sep-r}\quad \frac{\Delta_1 \vdash P_1 :: y : A \qquad \Delta_2 \vdash P_2 :: x : B}{\Delta_1 \mathbin{,} \Delta_2 \vdash \overline{x}[y].(P_1 \mid P_2) :: x : A * B}$$

$$\textsc{Sep-l}\quad \frac{\Gamma(x : B \mathbin{,} y : A) \vdash P :: z : C}{\Gamma(x : A * B) \vdash x(y).P :: z : C}$$

$$\textsc{Wand-r}\quad \frac{\Delta \mathbin{,} y : A \vdash P :: x : B}{\Delta \vdash x(y).P :: x : A \mathrel{-\!\!*} B}$$

$$\textsc{Wand-l}\quad \frac{\Delta \vdash P :: y : A \qquad \Gamma(x : B) \vdash Q :: z : C}{\Gamma(\Delta \mathbin{,} x : A \mathrel{-\!\!*} B) \vdash \overline{x}[y].(P \mid Q) :: z : C}$$

Typing: Additive Connectives

...just as additive conjunction and implication.

Conj-r

$$\frac{\Delta_1 \vdash P_1 :: y : A \qquad \Delta_2 \vdash P_2 :: x : B}{\Delta_1 \mathbin{;} \Delta_2 \vdash \overline{x}[y].(P_1 \mid P_2) :: x : A \wedge B}$$

Conj-l

$$\frac{\Gamma(x : B \mathbin{;} y : A) \vdash P :: z : C}{\Gamma(x : A \wedge B) \vdash x(y).P :: z : C}$$

Impl-r

$$\frac{\Delta \mathbin{;} y : A \vdash P :: x : B}{\Delta \vdash x(y).P :: x : A \to B}$$

Impl-l

$$\frac{\Delta \vdash P :: y : A \qquad \Gamma(x : B) \vdash Q :: z : C}{\Gamma(\Delta \mathbin{;} x : A \to B) \vdash \overline{x}[y].(P \mid Q) :: z : C}$$

Typing: Structural Rules

A fair/naive attempt at typing the spawn prefix:

WEAKENING

$$\frac{\Gamma(\Delta_2) \vdash P :: z : C}{\Gamma(\Delta_1 \,;\, \Delta_2) \vdash \boldsymbol{\rho}[x \mapsto \emptyset \mid x \in \Delta_1].P :: z : C}$$

CONTRACTION

$$\frac{\Gamma(\Delta^{(1)} \,;\, \Delta^{(2)}) \vdash P :: z : C}{\Gamma(\Delta) \vdash \boldsymbol{\rho}[x \mapsto x_1, x_2 \mid x \in \Delta].P :: z : C}$$

Typing: Structural Rules

A fair/naive attempt at typing the spawn prefix:

Weakening

$$\frac{\Gamma(\Delta_2) \vdash P :: z : C}{\Gamma(\Delta_1 \,\boldsymbol{;}\, \Delta_2) \vdash \boldsymbol{\rho}[x \mapsto \emptyset \mid x \in \Delta_1].P :: z : C}$$

Contraction

$$\frac{\Gamma(\Delta^{(1)} \,\boldsymbol{;}\, \Delta^{(2)}) \vdash P :: z : C}{\Gamma(\Delta) \vdash \boldsymbol{\rho}[x \mapsto x_1, x_2 \mid x \in \Delta].P :: z : C}$$

Unfortunately, this is not general enough for establishing type preservation.

Typing: Structural Rules

We need the following general rule

$$\textsc{Struct}\quad \frac{\Delta_2 \vdash P :: z : C \qquad \sigma : \Delta_1 \rightsquigarrow \Delta_2}{\Delta_1 \vdash \boldsymbol{\rho}[\sigma].P :: z : C}$$

where $\sigma : \Delta_1 \rightsquigarrow \Delta_2$ is a relation on bunches:
it captures transformations enacted by weakening, contraction, and merge.

Typing: Spawn Binding

We define $\sigma : \Delta_1 \rightsquigarrow \Delta_2$ as follows:

SPAWN-CONTRACT

$$[x \mapsto \{x_1, \ldots, x_n\} \mid x \in \Delta] : \Gamma(\Delta) \rightsquigarrow \Gamma\big(\Delta^{(1)} \mathbin{;} \cdots \mathbin{;} \Delta^{(n)}\big)$$

SPAWN-WEAKEN

$$[x \mapsto \emptyset \mid x \in \Delta_1] : \Gamma(\Delta_1 \mathbin{;} \Delta_2) \rightsquigarrow \Gamma(\Delta_2)$$

SPAWN-MERGE

$$\frac{\sigma_1 : \Delta_0 \rightsquigarrow \Delta_1 \qquad \sigma_2 : \Delta_1 \rightsquigarrow \Delta_2}{(\sigma_1 \ltimes \sigma_2) : \Delta_0 \rightsquigarrow \Delta_2}$$

Properties

We show the following meta-theoretical results:

Type preservation (protocols are respected)

Deadlock-freedom:
If a process $P :: x : A$ cannot reduce then it has an action on x

Weak normalizationx (proof by a combinatorial argument)

These properties confirm that our typing system defines a Curry-Howard interpretation for the BI sequent calculus

Closing

Closing Remarks

A new concurrent interpretation of BI in the spirit of Propositions-as-Sessions

- A new programming calculus, πBI, whose design and formulation are induced by structural principles in BI.
- It is reassuring (and encouraging) that the interpretation enjoys all results from propositions-as-sessions.

Closing Remarks

A new concurrent interpretation of BI in the spirit of Propositions-as-Sessions

- A new programming calculus, πBI, whose design and formulation are induced by structural principles in BI.
- It is reassuring (and encouraging) that the interpretation enjoys all results from propositions-as-sessions.
- Many other technical results in the paper, in particular about the coexistence of $A \wedge B$ and $A * B$ as protocols.

Closing Remarks

A new concurrent interpretation of BI in the spirit of Propositions-as-Sessions

- A new programming calculus, πBI, whose design and formulation are induced by structural principles in BI.
- It is reassuring (and encouraging) that the interpretation enjoys all results from propositions-as-sessions.
- Many other technical results in the paper, in particular about the coexistence of $A \wedge B$ and $A * B$ as protocols.
- LL and BI rely on very different semantic models; current work aims at new explanations of those differences.

A Bunch of Sessions:
Session Types beyond Linear Logic

Jorge A. Pérez (University of Groningen, The Netherlands)

joint work with
Emanuele D'Osualdo (MPI-SWS), Dan Frumin (Aarhus), and Bas van den Heuvel (Groningen)

UNIFYING
C•RRECTNESS FOR
C•MMUNICATING
S•FTWARE

June 19, 2023

More on the Unusual Example

$$
\dfrac{
  \dfrac{
    a_1 : A \vdash [a_1' \leftarrow a_1] :: a_1' : A
    \qquad
    \dfrac{
      a_2 : A \vdash [a_2' \leftarrow a_2] :: a_2' : A
      \qquad
      y : B \vdash [z \leftarrow y] :: z : B
    }{
      a_2 : A \,;\, y : A \to B \vdash \overline{y}[a_2'].([a_2' \leftarrow a_2] \mid [z \leftarrow y]) :: z : B
    }
  }{
    \dfrac{
      a_1 : A \,;\, a_2 : A \,;\, y : A \to A \to B \vdash \overline{y}[a_1'].\big([a_1' \leftarrow a_1] \mid \overline{y}[a_2'].(\dots)\big) :: z : B
    }{
      \dfrac{
        a : A \,;\, y : A \to A \to B \vdash \boldsymbol{\rho}[a \mapsto a_1, a_2].\big(\overline{y}[a_1'].(\dots)\big) :: z : B
      }{
        a : A \vdash z(y).\boldsymbol{\rho}[a \mapsto a_1, a_2].(\dots) :: z : (A \to A \to B) \to B
      }
    }
  }
}{
  \emptyset_{\mathsf{m}} \vdash z(a).z(y).\boldsymbol{\rho}[a \mapsto a_1, a_2].(\dots) :: z : A \mathrel{-\!\!*} (A \to A \to B) \to B
}
$$

Reduction Strategy for WN

Intuition: A spawn prefix is “smaller” than another one if it is closer to the top-level.

> If a process can perform a communication reduction or a forwarder reduction, then we do that reduction.
>
> Otherwise, if a process can only perform a reduction that involves a spawn prefix, then we
>
> > - select (an active) spawn prefix with the least depth;
> > - perform the spawn reduction;
> > - propagate the newly created spawn prefix to the very top-level, merging it with other spawn prefixes along the way.

Observational and Denotational Equivalence

Observational equivalence

Definition (Observational equivalence)

$\Delta \vdash P \simeq_o Q :: x : A$ if for any well-typed context $\mathcal{C}$, we have $\mathcal{C}[P] \Downarrow_\alpha \Leftrightarrow \mathcal{C}[Q] \Downarrow_\alpha$

Observational equivalence

Definition (Observational equivalence)

$\Delta \vdash P \simeq_o Q :: x : A$ if for any well-typed context $\mathcal{C}$, we have $\mathcal{C}[P] \Downarrow_\alpha \Leftrightarrow \mathcal{C}[Q] \Downarrow_\alpha$

$\boxed{\Delta \vdash P :: x : A \Rightarrow \Sigma \vdash \mathcal{C}[P] :: z : B \text{ where } \Sigma \text{ is closed}}$

Observational equivalence

Definition (Observational equivalence)

$\Delta \vdash P \simeq_o Q :: x : A$ if for any well-typed context $\mathcal{C}$, we have $\mathcal{C}[P] \Downarrow_\alpha \Leftrightarrow \mathcal{C}[Q] \Downarrow_\alpha$

Can we give a *local* characterization of observational equivalence?

Denotational Semantics

Fix a set Tag of *primitive tags* and a sequence of *atomic types* $\mathfrak{a}_1, \mathfrak{a}_2, \ldots$.

The interpretation $[\![A]\!] : \wp(\mathsf{Tag}) \to \mathsf{Set}$:

$$
\begin{aligned}
[\![1_m]\!](D) = [\![1_a]\!](D) &= \{*\} \\
[\![A \vee B]\!](D) &= [\![A]\!](D) + [\![B]\!](D) \\
[\![A \wedge B]\!](D) &= [\![A]\!](D) \times [\![B]\!](D) \\
[\![A * B]\!](D) &= \Sigma_{D = D_1 \cup D_2, D_1 \perp D_2}.\, [\![A]\!](D_1) \times [\![B]\!](D_2) \\
[\![A \to B]\!](D) &= [\![A]\!](D) \to [\![B]\!](D) \\
[\![A \mathrel{-\!\!*} B]\!](D) &= \Pi_{D' \perp D}.\, [\![A]\!](D') \to [\![B]\!](D \cup D')
\end{aligned}
$$

Provenance: In a process with a session $A * B$ the subprocesses implementing A and B have a different origin.

Denotational Semantics: Properties

A process is interpreted as a function polymorphic in a set of tags D:

$$\Delta \vdash P :: x : A \Rightarrow [\![P]\!] : \forall D, [\![\Delta]\!](D) \to [\![A]\!](D)$$

Gives a compositional local semantics based on doubly-closed Cartesian categories.

Properties:

$$P \equiv Q \Rightarrow [\![P]\!] = [\![Q]\!];$$

Denotational Semantics: Properties

A process is interpreted as a function polymorphic in a set of tags D:

$$\Delta \vdash P :: x : A \Rightarrow [\![P]\!] : \forall D, [\![\Delta]\!](D) \rightarrow [\![A]\!](D)$$

Gives a compositional local semantics based on doubly-closed Cartesian categories.

Properties:

$$P \equiv Q \Rightarrow [\![P]\!] = [\![Q]\!];$$

$$P \longrightarrow Q \Rightarrow [\![P]\!] = [\![Q]\!];$$

Denotational Semantics: Properties

A process is interpreted as a function polymorphic in a set of tags D:

$$\Delta \vdash P :: x : A \Rightarrow [\![P]\!] : \forall D, [\![\Delta]\!](D) \to [\![A]\!](D)$$

Gives a compositional local semantics based on doubly-closed Cartesian categories.

Properties:

$$P \equiv Q \Rightarrow [\![P]\!] = [\![Q]\!];$$

$$P \longrightarrow Q \Rightarrow [\![P]\!] = [\![Q]\!];$$

$$[\![P]\!] = [\![Q]\!] \Rightarrow P \simeq_o Q$$