

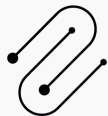
A Bunch of Sessions:

Session Types beyond Linear Logic

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

joint work with

Emanuele D'Ossualdo (MPI-SWS), Dan Frumin (Aarhus), and Bas van den Heuvel (Groningen)



UNIFYING
C•RRECTNESS FOR
C•MMUNICATING
S•FTWARE

June 18, 2023

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.

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 normalizing.
- ✓ 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

$$\frac{}{\Gamma, A \vdash A}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$$

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Sequential interpretations

Intuitionistic Logic \leftrightarrow Simply-typed λ -calculus

$$\frac{}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : t : A \rightarrow B}$$

$$\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B}$$

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:

$$\begin{array}{c} \frac{}{A \vdash A} \qquad \frac{\Delta_1 \vdash A \quad \Delta_2, A \vdash C}{\Delta_1, \Delta_2 \vdash C} \\[2ex] \frac{\Delta, A, B \vdash C}{\Delta, A \otimes B \vdash C} \qquad \frac{\Delta_1 \vdash A \quad \Delta_2 \vdash B}{\Delta_1, \Delta_2 \vdash A \otimes B} \end{array}$$

Curry-Howard correspondences

Extracting a well-behaved programming language from a logic.

Concurrent interpretations

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

Sample rules:

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

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 π BI \leftarrow THIS WORK

Logic of Bunched Implications

Bunched Implications (BI)

BI combines intuitionistic logic and (multiplicative) linear logic:

$$A, B \in Prop ::= 1_a \mid A \vee B \mid A \wedge B \mid A \rightarrow B$$

Bunched Implications (BI)

BI combines intuitionistic logic and (multiplicative) linear logic:

$$A, B \in Prop ::= \mathbf{1}_a \mid A \vee B \mid A \wedge B \mid A \rightarrow B \\ \mid \mathbf{1}_m \mid A * B \mid A \multimap B$$



Intuitionistic logic

Bunched Implications (BI)

BI combines intuitionistic logic and (multiplicative) linear logic:

$$A, B \in Prop ::= \mathbf{1}_a \mid A \vee B \mid A \wedge B \mid A \rightarrow B \\ \mid \mathbf{1}_m \mid A * B \mid A \multimap B$$


Linear logic (fragment)

Bunched Implications (BI)

BI combines intuitionistic logic and (multiplicative) linear logic:

$$A, B \in \mathbf{Prop} ::= \mathbf{1}_a \mid A \vee B \mid A \wedge B \mid A \rightarrow B \\ \mid \mathbf{1}_m \mid A * B \mid A \multimap B$$

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)

$$A \wedge (A \rightarrow B) \rightarrow B$$

$$A * (A \multimap B) \multimap B$$

Bunched Implications (BI)

$$A \wedge (A \rightarrow B) \rightarrow B$$

$$A \rightarrow A \wedge A$$

$$A \wedge B \rightarrow A$$

$$A * (A \multimap B) \multimap B$$

$$A \not\multimap A * A$$

$$A * B \not\multimap A$$

Bunched Implications (BI)

$$A \wedge (A \rightarrow B) \rightarrow B$$

$$A \rightarrow A \wedge A$$

$$A \wedge B \rightarrow A$$

$$A \wedge (A \rightarrow B) \rightarrow A \wedge B$$

$$A * (A \multimap B) \multimap B$$

$$A \not\multimap A * A$$

$$A * B \not\multimap A$$

$$A * (A \multimap B) \not\multimap A * B$$

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


Sequent: $\Gamma \vdash \phi$

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

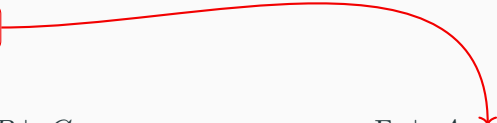
$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1; \Gamma_2 \vdash A \wedge B}$$

A sequent calculus for BI

Left and right rules


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

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


$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1; \Gamma_2 \vdash A \wedge B}$$

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

A sequent calculus for BI

Structural rules

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

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

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1; \Gamma_2 \vdash A \wedge B}$$

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

A sequent calculus for BI

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

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A * B}$$

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

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1; \Gamma_2 \vdash A \wedge B}$$

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

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

A sequent calculus for BI

$$\Gamma ::= A \mid \Gamma ; \Gamma \mid \Gamma , \Gamma \mid \dots$$

$$\frac{\Delta(A, B) \vdash C}{\Delta(A * B) \vdash C}$$

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A * B}$$

$$\frac{\Delta(A ; B) \vdash C}{\Delta(A \wedge B) \vdash C}$$

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1 ; \Gamma_2 \vdash A \wedge B}$$

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

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

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

The Process Language

The π BI Process Language

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

The Spawn Construct: Intuitions

A prefix defined hand-in-hand with structural properties in BI.

We write

$$\rho[\sigma].P$$

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 defined hand-in-hand with structural properties in BI.

We write

$$\rho[\sigma].P$$

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.]

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

Reduction Semantics

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

RED-COMM-R

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

RED-UNIT-R

$$(\nu x). (x().Q \mid_x \bar{x}\langle \rangle) \longrightarrow Q$$

RED-CASE

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

RED-FWD-R

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

Reduction Semantics

Reduction for spawn is more interesting. Consider duplication:

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

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

Reduction Semantics

Reduction for spawn is more interesting. Consider duplication:

$$\begin{aligned} (\nu x).(P \mid_x \rho[x \mapsto x_1, x_2].Q) &\longrightarrow \\ (\nu x_1).(P[x_1/x] \mid_{x_1} (\nu x_2).(P[x_2/x] \mid_{x_2} Q)), \end{aligned}$$

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

Now consider process discarding:

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

If P only communicates through x .

Reductions

Let's gradually get more precise:

RED-SPAWN*

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

Reductions

Let's gradually get more precise:

RED-SPAWN*

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

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*

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

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*

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

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...

The general rule:

RED-SPAWN

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

The general rule:

RED-SPAWN

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

RED-SPAWN-R

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

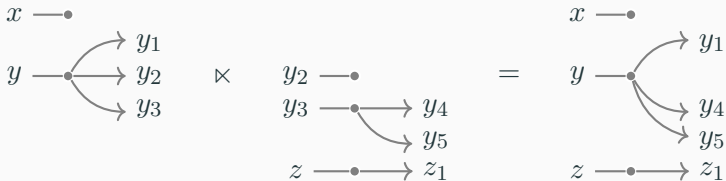
Reduction Semantics

We can “accumulate” spawn prefixes:

RED-SPAWN-MERGE

$$\rho[\sigma_1].\rho[\sigma_2].P \longrightarrow \rho[\sigma_1 \ltimes \sigma_2].P$$

where \ltimes is the **merge operator**. It operates as follows:



Congruences (Selection)

CONG-ASSOC-L

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

CONGR-ASSOC-R

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

CONGR-SPAWN-SWAP

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

An Unusual Example

Consider the following π BI process:

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

Intuitively:

Process P receives a single session of type A over z through a *multiplicative* input (to be typed with \multimap).

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 π BI process:

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

Intuitively:

Process P receives a single session of type A over z through a *multiplicative* input (to be typed with \multimap).

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

$$\emptyset_m \vdash P :: z : A \multimap (A \rightarrow A \rightarrow B) \rightarrow B$$

BI as a Type System for Sessions

Typing: Identity and Cut

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

Identity as forwarding:

FWD

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

Cut as composition:

CUT

$$\frac{\Delta \vdash P :: x : A \quad \Gamma(x : A) \vdash Q :: z : C}{\Gamma(\Delta) \vdash (\nu x).(P \mid Q) :: z : C}$$

Typing: Multiplicative Connectives

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

SEP-R

$$\frac{\Delta_1 \vdash P_1 :: y : A \quad \Delta_2 \vdash P_2 :: x : B}{\Delta_1, \Delta_2 \vdash \bar{x}[y].(P_1 \mid P_2) :: x : A * B}$$

SEP-L

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

WAND-R

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

WAND-L

$$\frac{\Delta \vdash P :: y : A \quad \Gamma(x : B) \vdash Q :: z : C}{\Gamma(\Delta, x : A \multimap B) \vdash \bar{x}[y].(P \mid Q) :: z : C}$$

Typing: Additive Connectives

...just as additive conjunction and implication.

CONJ-R

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

CONJ-L

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

IMPL-R

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

IMPL-L

$$\frac{\Delta \vdash P :: y : A \quad \Gamma(x : B) \vdash Q :: z : C}{\Gamma(\Delta ; x : A \rightarrow B) \vdash \bar{x}[y].(P \mid Q) :: z : C}$$

Typing: Structural Rules

A fair/naive attempt at typing the spawn prefix:

WEAKENING

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

CONTRACTION

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

Typing: Structural Rules

A fair/naive attempt at typing the spawn prefix:

WEAKENING

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

CONTRACTION

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

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

We need the following general rule

STRUCT

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

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

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

SPAWN-WEAKEN

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

SPAWN-MERGE

$$\frac{\sigma_1: \Delta_0 \rightsquigarrow \Delta_1 \quad \sigma_2: \Delta_1 \rightsquigarrow \Delta_2}{(\sigma_1 \times \sigma_2): \Delta_0 \rightsquigarrow \Delta_2}$$

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 normalization (proof by a combinatorial argument)

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

Closing

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.

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.

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.

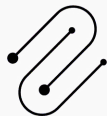
A Bunch of Sessions:

Session Types beyond Linear Logic

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

joint work with

Emanuele D'Ossualdo (MPI-SWS), Dan Frumin (Aarhus), and Bas van den Heuvel (Groningen)



UNIFYING
C•ORRECTNESS FOR
C•MMUNICATING
S•FTWARE

June 18, 2023

More on the Unusual Example

$$\begin{array}{c}
 \frac{a_2 : A \vdash [a'_2 \leftarrow a_2] :: a'_2 : A \quad y : B \vdash [z \leftarrow y] :: z : B}{a_1 : A \vdash [a'_1 \leftarrow a_1] :: a'_1 : A \quad a_2 : A; y : A \rightarrow B \vdash \bar{y}[a'_2].([a'_2 \leftarrow a_2] \mid [z \leftarrow y]) :: z : B} \\
 \hline
 a_1 : A; a_2 : A; y : A \rightarrow A \rightarrow B \vdash \bar{y}[a'_1].([a'_1 \leftarrow a_1] \mid \bar{y}[a'_2].(\dots)) :: z : B \\
 \hline
 a : A; y : A \rightarrow A \rightarrow B \vdash \rho[a \mapsto a_1, a_2].(\bar{y}[a'_1].(\dots)) :: z : B \\
 \hline
 a : A \vdash z(y).\rho[a \mapsto a_1, a_2].(\dots) :: z : (A \rightarrow A \rightarrow B) \rightarrow B \\
 \hline
 \emptyset_m \vdash z(a).z(y).\rho[a \mapsto a_1, a_2].(\dots) :: z : A \multimap (A \rightarrow A \rightarrow B) \rightarrow B
 \end{array}$$

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 $\mathcal{C}[P] \Downarrow_\alpha \Leftrightarrow \mathcal{C}[Q] \Downarrow_\alpha$



$\Delta \vdash P :: x : A \Rightarrow \Sigma \vdash \mathcal{C}[P] :: z : B$ where Σ 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* $\mathfrak{a}_1, \mathfrak{a}_2, \dots$.

The interpretation $\llbracket A \rrbracket : \wp(\text{Tag}) \rightarrow \text{Set}$:

$$\llbracket 1_m \rrbracket(D) = \llbracket 1_a \rrbracket(D) = \{*\}$$

$$\llbracket A \vee B \rrbracket(D) = \llbracket A \rrbracket(D) + \llbracket B \rrbracket(D)$$

$$\llbracket A \wedge B \rrbracket(D) = \llbracket A \rrbracket(D) \times \llbracket B \rrbracket(D)$$

$$\llbracket A * B \rrbracket(D) = \Sigma_{D=D_1 \cup D_2, D_1 \perp D_2} \cdot \llbracket A \rrbracket(D_1) \times \llbracket B \rrbracket(D_2)$$

$$\llbracket A \rightarrow B \rrbracket(D) = \llbracket A \rrbracket(D) \rightarrow \llbracket B \rrbracket(D)$$

$$\llbracket A \multimap B \rrbracket(D) = \Pi_{D' \perp D} \cdot \llbracket A \rrbracket(D') \rightarrow \llbracket B \rrbracket(D \cup D')$$

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 :

$$\Delta \vdash P :: x : A \Rightarrow \llbracket P \rrbracket : \forall D, \llbracket \Delta \rrbracket(D) \rightarrow \llbracket A \rrbracket(D)$$

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

Properties:

$$P \equiv Q \Rightarrow \llbracket P \rrbracket = \llbracket Q \rrbracket;$$

Denotational Semantics: Properties

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

$$\Delta \vdash P :: x : A \Rightarrow \llbracket P \rrbracket : \forall D, \llbracket \Delta \rrbracket(D) \rightarrow \llbracket A \rrbracket(D)$$

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

Properties:

$$P \equiv Q \Rightarrow \llbracket P \rrbracket = \llbracket Q \rrbracket;$$

$$P \longrightarrow Q \Rightarrow \llbracket P \rrbracket = \llbracket Q \rrbracket;$$

Denotational Semantics: Properties

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

$$\Delta \vdash P :: x : A \Rightarrow \llbracket P \rrbracket : \forall D, \llbracket \Delta \rrbracket(D) \rightarrow \llbracket A \rrbracket(D)$$

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

Properties:

$$P \equiv Q \Rightarrow \llbracket P \rrbracket = \llbracket Q \rrbracket;$$

$$P \longrightarrow Q \Rightarrow \llbracket P \rrbracket = \llbracket Q \rrbracket;$$

$$\llbracket P \rrbracket = \llbracket Q \rrbracket \Rightarrow P \simeq_o Q$$