Linear Logic: A Logical Foundation for Concurrent Computation

Bernardo Toninho
[joint work with Luís Caires, Frank Pfenning, Jorge Pérez and Henry DeYoung]

Computer Science Department
Carnegie Mellon University
Universidade Nova de Lisboa

ENS, Lyon
Oct. 11, 2012
Introduction
Linear Logic and Concurrency

Linear Logic [Girard 1987]
- A marriage of classical dualities and constructivism.
- A logic of resources and interaction.
- Resource independence captures non-determinism/concurrency.
- Linear logic as the logic of concurrency?

Linear Logic and Concurrency
Initial efforts explored the connections to concurrency:
- Abramsky’s computational interpretation [Abramsky 93]
- Bellin and Scott’s refinement to a $\pi$-calculus [BellinScott 94]
- Research shifted to more geometric approaches.
- No real “Curry-Howard interpretation”.

Bernardo Toninho (CMU & UNL)
Introduction
Why logic and concurrency?

Why does it matter?
- *Everything* is moving towards concurrency and distribution.
- Building these systems turns out to be really hard.
- Reasoning about these systems is even harder.

Why does a logical foundation help?
- Logic is well understood.
- Reasoning built-in.
- Good metalogical properties map to good program properties.

Curry-Howard Isomorphism
- Sequential: Intuitionistic Logic $\simeq \lambda$-calculus
- Concurrency: Linear Logic $\simeq$ ???
Introduction
A Concurrency Theoretic Approach

Structuring Communication

- Communication without structure is hard to reason about.
- Structure communication around the concept of a session.
- Predetermined sequences of interactions along a (session) channel:
  - “Input a number, output a string and terminate.”
  - “Either output or input a number.”

Sessions

- Specify communication behavior as sessions.
- Check that programs adhere to specification (session fidelity).
Session Types [Honda93]
- Types are descriptions of communication behavior.
- A way of guaranteeing communication discipline, statically.

Session Types and ILL [CairesPfenning01]
- Its possible to interpret session types and linear logic propositions.
- Linear logic proofs as process typing derivations.
- Proof dynamics as process dynamics.

Why is this important?
- Results from logic carry over to session typings:
  - Progress, session fidelity, type preservation “for free”.
- Extending the system with ideas from logic becomes possible.
Research Questions

- Can we use this as a logical theory for (session-based) concurrency?
- A logical understanding of phenomena in concurrency?
- Mapping logical phenomena to concurrency?
- Does it help us to reason about concurrency?

Roadmap

- Proof Conversions, Type Isomorphisms and Process Equivalence
- Asynchronous Communication
- Concurrent Evaluation Strategies
- Richer Type Structures
Key Ideas

- Session Types as Intuitionistic Linear Propositions.
- Sequent calculus rules as $\pi$-calculus typing rules.
- Cut reduction as process reduction.

Why sequent calculus?

- Duality of offering (right rules) and using (left rules) a session.
- Proof composition (cut) as process composition.
- Identity as forwarding/renaming.
Typing Judgment

\[ \Gamma \vdash u_1 : A_1, \ldots, u_m : A_m ; x_1 : A_1, \ldots, x_n : A_n \Rightarrow P :: x : A \]

Process \( P \) provides \( A \) along \( x \) if composed with sessions in \( \Delta \) and \( \Gamma \).

Cut as Composition

\[ \Gamma ; \Delta \Rightarrow P :: x : A \quad \Gamma ; \Delta', x : A \Rightarrow Q :: z : C \]

\[ \Gamma ; \Delta, \Delta' \Rightarrow (\forall x)(P \mid Q) :: z : C \]

Parallel composition of \( P \), offering \( x : A \) and \( Q \), using \( x : A \).

Identity as Renaming

\[ \Gamma ; x : A \Rightarrow [x \leftrightarrow z] :: z : A \]
**Context**

Caires-Pfenning Interpretation - Propositions

---

**Multiplicative Conjunction**

\[
\frac{\Gamma; \Delta \rightsquigarrow P_1 :: y:A \quad \Gamma; \Delta' \rightsquigarrow P_2 :: x:B}{\Gamma; \Delta, \Delta' \rightsquigarrow (\nu y)x\langle y \rangle.(P_1 \mid P_2) :: x:A \otimes B}
\]

\[
\frac{\Gamma; \Delta, y : A, x : B \rightsquigarrow Q :: z:C}{\Gamma; \Delta, x:A \otimes B \rightsquigarrow x(y).Q :: z:C}
\]

---

**Proof Reduction**

\[
\frac{\Gamma; \Delta, \Delta' \rightsquigarrow (\nu x)((\nu y)x\langle y \rangle.(P_1 \mid P_2) \mid x(y).Q) :: z:C}{\Gamma; \Delta, \Delta' \rightsquigarrow (\nu x)(\nu y)(P_1 \mid P_2 \mid Q) :: z:C}
\]
Linear Implication

\[
\frac{\Gamma; \Delta, y : A \Rightarrow P :: x:B}{\Gamma; \Delta \Rightarrow x(y).P :: x:A \rightarrow B}
\]

\[
\frac{\Gamma; \Delta \Rightarrow Q_1 :: y:A \quad \Gamma; \Delta', x:B \Rightarrow Q_2 :: z:C}{\Gamma; \Delta, \Delta', x:A \rightarrow B \Rightarrow (\nu y)x\langle y\rangle.(Q_1 | Q_2) :: z:C}
\]

Linear Implication as input. Reduction is the same as for \(\otimes\).
### Multiplicative Unit

\[ \frac{\Gamma; \cdot \Rightarrow 0 :: x : 1}{1R} \]

\[ \frac{\Gamma; \Delta \Rightarrow Q :: z : C}{1L} \]

\[ \Gamma; \Delta \Rightarrow (\nu x)(0 | Q) :: z : C \]

\[ \equiv \Gamma; \Delta \Rightarrow Q :: z : C \]

### Proof Reduction

**Multiplicative Unit as Termination**
Additive Conjunction

\[ \Gamma; \Delta \rightarrow P_1 :: x:A \quad \Gamma; \Delta \rightarrow P_2 :: x:B \]
\[ \Gamma; \Delta \rightarrow x.\text{case}(P_1, P_2) :: x:A \& B \]

\[ \Gamma; \Delta, x:A \rightarrow Q :: z:C \]
\[ \Gamma; \Delta, x:A \& B \rightarrow x.\text{inl}; Q :: z:C \]

&L₁

&R

Proof Reduction

\[ \Gamma; \Delta, \Delta' \rightarrow (\nu x)(x.\text{case}(P_1, P_2) \mid x.\text{inl}; Q) :: z:C \]
\[ \rightarrow \Gamma; \Delta, \Delta' \rightarrow (\nu x)(P_1 \mid Q) :: z:C \]
Additive Disjunction

\[
\begin{align*}
\Gamma; \Delta \quad &\quad P : : x : A \\
\Gamma; \Delta \quad &\quad x \cdot \text{inl} ; \quad P : : x : A \oplus B \quad \oplus R_1 \\
\Gamma; \Delta, x : A \quad &\quad Q_1 : : z : C \\
\Gamma; \Delta, x : B \quad &\quad Q_2 : : z : C \quad \oplus L \\
\Gamma; \Delta, x : A \oplus B \quad &\quad x \cdot \text{case}(Q_1, Q_2) : : z : C
\end{align*}
\]

Same proof reductions as &.
Persistent Cut

\[
\begin{align*}
\Gamma; \cdot &\implies P :: x:A & \Gamma, u:A; \Delta &\implies Q :: z:C \\
\Gamma; \Delta &\implies (\nu u)(!u(x).P \mid Q) :: z:C
\end{align*}
\]

Parallel composition of \( P \), offering \( x:A \) and \( Q \), using \( u:A \) persistently.

Copy

\[
\begin{align*}
\Gamma, u:A; \Delta, x:A &\implies P :: z:C \\
\Gamma, u:A; \Delta &\implies (\nu x)u\langle x\rangle.Q :: z:C
\end{align*}
\]

Proof Reduction

\[
\begin{align*}
\Gamma; \Delta &\implies (\nu u)(!u(x).P \mid (\nu x)u\langle x\rangle.Q) :: z:C \\
\rightarrow \Gamma; \Delta &\implies (\nu u)(!u(x).P \mid (\nu x)(P \mid Q)) :: z:C
\end{align*}
\]
Exponential

\[
\begin{align*}
\Gamma; \cdot & \implies P :: y:A \\
\Gamma; \cdot & \implies !x(y).P :: x!:A \\
\Gamma; \cdot & \implies P :: y:A
\end{align*}
\]

\[
\begin{align*}
\Gamma, u:A; \Delta & \implies P :: z:C \\
\Gamma, u:A; \Delta, x!:A & \implies P\{x/u\} :: z:C
\end{align*}
\]

Proof reduction transforms a cut into a cut\(^1\) (struct. equivalence).
Operational Correspondence and Subject Reduction

If $\Gamma; \Delta \vdash P :: z:A$ and $P \rightarrow P'$ then $\exists Q$ such that $\Gamma; \Delta \vdash Q :: z:A$ and $P' \equiv Q$.

Global Progress

\[
live(P) \triangleq (\nu x)(Q | R) \text{ with } Q \equiv \pi.Q' \text{ or } Q \equiv [x \leftrightarrow y]\]

If $\vdash P :: x:1$ and $live(P)$ then $\exists Q$ such that $P \rightarrow Q$. 
To summarize this interpretation:

- Linear Propositions as Session Types.
- Intuitionistic sequent proofs as session-typed processes.
- Process reduction maps to proof conversion.
- . . . but not all proof conversions are process reductions!
Proof Conversions and Type Isomorphisms

Introduction

Proof Conversions

- Process reductions map to principal cut reductions.
- What about the remaining proof conversions?
- Can we understand them in concurrency theoretic terms?

Approach

We decompose proof conversions into three classes:

- Computational Conversions (i.e. principal cut conversions).
- Cut Conversions (i.e. permutting two cuts in a proof).
- Commuting Conversions (i.e. commuting inference rules).

First two correspond to reductions and structural equivalences.
Commuting Conversions induce a congruence $\cong_c$ on typed processes.

$\otimes L / \otimes L$ Commuting Conversion

$x : A \otimes B, z : C \otimes D \implies x(y).z(w).P \cong_c z(w).x(y).P :: v:E$

Commuting (input) prefixes appears, at first, counterintuitive.

Typed Contextual Equivalence

In any well-typed context, we cannot distinguish the two processes:

$\nu x) \nu z)(x(y).z(w).P | R_x | S_z) \cong (\nu x)(\nu z)(z(w).x(y).P | R_x | S_z) :: v:E$

Actions along $x$ and $z$ are not observable.
Typed Contextual Equivalence

How to define this equivalence in a tractable way?

Typed Contextual Bisimilarity.

Contextual Bisimilarity

Contextual: For all typed contexts...

Typed bisimilarity on closed processes:

\[ P \sim Q :: x:A \rightarrow B \text{ iff } P \xrightarrow{x(y)} P' \text{ implies } Q \xrightarrow{x(y)} Q' \text{ and } \forall R. \Rightarrow R :: y:A \text{ we have } (\nu y)(P | R) \sim (\nu y)(Q | R) :: x:B \]

\[ P \sim Q :: x:C \text{ iff } P \xrightarrow{\tau} P' \text{ implies } Q \Rightarrow Q' \text{ and } P' \sim Q' :: x:C. \]

\[ \ldots \]
Proof Conversions

Issue

\( \otimes \mathit{L}/\otimes \mathit{L} \) Conversion Revisited

\[(\nu x)(\nu z)(x(y).z(w).P \mid R_x \mid S_z) \sim (\nu x)(\nu z)(z(w).x(y).P \mid R_x \mid S_z) :: \nu : E?\]

- Suppose input along \( x \) matches an output in \( R_x \) on the left proc.
- How do we know the right side process can match it?
- What if \( S_z \) never outputs to \( z \)?
- Requires *termination*!

Termination and Bisimilarity

- Can we develop a uniform solution?
- Inspiration from functional “world”: Linear Logical Relations!

Bernardo Toninho (CMU & UNL)
Linear Logical Relations [Pérez et al. 12]

- Termination: Inductively defined unary predicate.
- Contextual Bisimulation: Co-inductively defined binary relation.

Logical Predicate

- Terminating by construction.
- Inductive on typing derivations: $\mathcal{L}[\Gamma; \Delta \vdash T]$
  - $P \in \mathcal{L}[\Gamma; y:A, \Delta \vdash T]$ if $\forall R \in \mathcal{L}[y:A].(\nu y)(R \mid P) \in \mathcal{L}[\Gamma; \Delta \vdash T]$.
  - Base case is inductive on types:
    - $P \in \mathcal{L}[z:A \rightarrow B] \triangleq$ if $P \xrightarrow{z(y)} P'$ then $\forall Q \in \mathcal{L}[y:A].(\nu y)(P' \mid Q) \in \mathcal{L}[z:B]$
    - $\ldots$

Typing implies Termination

If $\Gamma; \Delta \vdash P :: T$ then $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$. 
Typed Bisimilarity

- Relational generalization of the predicate.
- Inductive on typing derivations: $\Gamma; \Delta \vdash \mathcal{P} \mathcal{R} \mathcal{Q} :: T$
  - If $\Gamma; \Delta, y : A \vdash \mathcal{P} \mathcal{R} \mathcal{Q} :: T$ then $\forall R. \vdash R :: y : A$
  - $\Gamma; \Delta \vdash (\nu y)(\mathcal{P} \mathcal{R} \mathcal{Q} :: T)$.
- Base case is inductive/coinductive on types:
  - $\vdash \mathcal{P} \mathcal{R} \mathcal{Q} :: x : A \rightarrow B$ if $P \xrightarrow{x(y)} P'$ implies $Q \xrightarrow{x(y)} Q'$ and $\forall R. \vdash R :: y : A$
  - we have $\vdash (\nu y)(\mathcal{P} \mathcal{R} \mathcal{Q} :: x : B)$
  - …
- $\approx$ is the largest such relation.

Soundness of Commuting Conversions

If $\Gamma; \Delta \vdash P \approx_c Q :: T$ then $\Gamma; \Delta \vdash P \approx Q :: T$
Type Isomorphisms
Definition and Validation

Type Isomorphism \((A \simeq B)\)
Types \(A\) and \(B\) are iso. if there are proofs \(\pi_A\) of \(B \vdash A\) and \(\pi_B\) of \(A \vdash B\), composing in both direction to identity.

Session Type Isomorphisms \((A \simeq S B)\)
Session types \(A\) and \(B\) are iso. if there are processes \(P\) and \(Q\):
- \(x:A \vdash P :: y:B\) and \(y:B \vdash Q :: x:A\).
- \(x:A \vdash (\nu y)(P | Q) \approx [x \leftrightarrow z] :: z:A\).
- \(y:B \vdash (\nu x)(Q | P) \approx [y \leftrightarrow z] :: z:B\).

Validating Isomorphisms
If \(A \simeq B\) then \(A \simeq S B\).
Asynchronous Communication

Asynchrony

Asynchrony for the Concurrency Theorist
- A more realistic form of communication.
- More challenging to reason about.

Asynchrony for the Proof Theorist
- Eliminates some of the “bureaucracy of syntax”.
- The order in which certain proof rules are applied doesn’t matter.

Can we develop an asynchronous process assignment with the same good properties?
Asynchronous Communication

Process Assignment [DeYoung et al. 12]

- Uses the same rules, but slightly different processes.
- Input clauses stay basically unchanged:
  - Since input is binding, it's a synchronization point.
- Outputs are asynchronous: \( x\langle y\rangle . P \) vs. \( x\langle y\rangle | P \).

Asynchronous Assignment for \( \rightarrow \) – Tentative

\[
\Gamma; \Delta, y:A \vdash P :: x:B \\
\Gamma; \Delta \vdash x(y).P :: x:A \rightarrow B \quad \rightarrow R? \\
\Gamma; \Delta \vdash Q_1 :: y:A \quad \Gamma; \Delta', x:B \vdash Q_2 :: z:C \\
\Gamma; \Delta, \Delta', x:A \rightarrow B \vdash (\nu y)(x\langle y\rangle | Q_1 | Q_2) :: z:C \quad \rightarrow L?
\]
Asynchronous Communication

Process Assignment

Problem

Consider the type \( A_1 \rightarrow (A_2 \rightarrow B) \):

\[
\begin{align*}
\Delta_2 \vdash P_2 :: y_2 : A_2 & \quad \Delta_3, x : B \vdash Q :: z : C \\
\Delta_1 \vdash P_1 :: y_1 : A_1 & \quad \Delta_2, \Delta_3, x : A_2 \rightarrow B \vdash (\nu y_2)(x \langle y_2 \rangle | P_2 | Q) :: z : C \\
\Delta_1, \Delta_2, \Delta_3, x : A_1 \rightarrow (A_2 \rightarrow B) \vdash (\nu y_1)(x \langle y_1 \rangle | P_1 | (\nu y_2)(x \langle y_2 \rangle | P_2 | Q)) :: z : C
\end{align*}
\]

- Process listening on \( x \) expects \( A_1 \) followed by \( A_2 \).
- Asynchrony makes it that \( y_2 : A_2 \) can be received before \( y_1 : A_1 \).
- Inherently unsafe – \( A_1 \) and \( A_2 \) can be completely different types.

Fortunately, there’s an easy fix.
Asynchronous Communication

Asynchronous Assignment for \( \rightarrow \circ \) – Fixed

\[
\Gamma; \Delta \vdash Q_1 :: y:A \quad \Gamma; \Delta', x':B \vdash Q_2 :: z:C \\
\Gamma; \Delta, \Delta', x:A \rightarrow B \vdash (\nu y, x')(x\langle y, x' \rangle \mid Q_1 \mid Q_2) :: z:C
\]

\[
\Gamma; \Delta \vdash P :: x':B \\
\Gamma; \Delta \vdash x(y, x').P :: x:A \rightarrow B
\]

Proof Reduction

\[
\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\nu x)(x(y, x').P \mid (\nu y)(\nu x')(x\langle y, x' \rangle \mid Q_1 \mid Q_2)) :: z:C
\]

\[
\longrightarrow \Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\nu x')(((\nu y)(Q_1 \mid P) \mid Q_2)) :: z:C
\]

Standard asynchronous \( \pi \)-calculus reduction.
Asynchronous Communication

**Process Assignment**

### Asynchronous Assignment for \( \otimes \)

\[
\Gamma; \Delta \vdash P_1 :: y : A \quad \Gamma; \Delta' \vdash P_2 :: x': B
\]

\[
\Gamma; \Delta, \Delta' \vdash (\nu y, x')(x\langle y, x' \rangle \mid P_1 \mid P_2) :: x : A \otimes B \quad \otimes R
\]

\[
\Gamma; \Delta, y : A, x' : B \vdash Q :: z : C
\]

\[
\Gamma; \Delta, x : A \otimes B \vdash x(y, x').Q :: z : C \quad \otimes L
\]

### Asynchronous Assignment for 1

\[
\Gamma; \cdot \vdash x\langle \rangle :: x : 1
\]

\[
\Gamma; \Delta' \vdash Q :: z : C
\]

\[
\Gamma; \Delta', x : 1 \vdash x() . 0 \mid Q :: z : C \quad 1L
\]

### Proof Reduction

\[
\Gamma; \Delta \vdash (\nu x)(x\langle \rangle \mid x() . 0 \mid Q) :: z : C \quad \rightarrow \quad \Gamma; \Delta \vdash Q :: z : C
\]
Asynchronous Communication

Process Assignment

Asynchronous Assignment for \&

\[ \Gamma; \Delta \vdash P_1 :: x'_1 : A \quad \Gamma; \Delta \vdash P_2 :: x'_2 : B \]
\[ \Gamma; \Delta \vdash x\text{.case}((x'_1).P_1, (x'_2).P_2) :: x : A \& B \] \&R

\[ \Gamma; \Delta, x'_1 : A \vdash Q :: z : C \]
\[ \Gamma; \Delta, x : A \& B \vdash (\nu x'_1)(x\text{.inl}\langle x'_1 \rangle \mid Q) :: z : C \] \&L1

Proof Reduction

\[ \Gamma; \Delta \vdash (\nu x)(x\text{.case}((x'_1).P_1, (x'_2).P_2) \mid (\nu x'_1)(x\text{.inl}\langle x'_1 \rangle \mid Q)) :: z : C \]

\[ \rightarrow \Gamma; \Delta \vdash (\nu x'_1)(P_1 \mid Q) :: z : C \]
Asynchronous Assignment for 

\[
\Gamma, u:A; \Delta, x:A \vdash Q :: z:C \\
\frac{}{\Gamma, u:A; \Delta \vdash (\nu x)(u\langle x \rangle \mid Q) :: z:C}
\]

\[
\Gamma; \cdot \vdash P :: y:A \\
\frac{}{\Gamma; \cdot \vdash (\nu u)(x\langle u \rangle \mid !u(y).P) :: x:!A}
\]

\[
\frac{}{\Gamma \vdash (\nu u)(x\langle u \rangle \mid !u(y).P) \mid x(u).Q) :: z:C}
\]

Proof Reduction

\[
\Gamma; \Delta \vdash (\nu x)((\nu u)(x\langle u \rangle \mid !u(y).P) \mid x(u).Q) :: z:C
\]

\[
\rightarrow \Gamma; \Delta \vdash (\nu u)(!u(y).P \mid Q) :: z:C
\]
Asynchronous Communication

Metatheory

Proof Conversions Redux

- Previously, commuting conversions were all obs. equivalences.
- Asynchrony divides commuting conversions in two sub-classes:
  1. Output Conversions: Commuting two outputs, or output with cut.
  2. Input Conversions: Commuting some input.
- All conversions in 1 are now structural equivalences:
  \[
  (\nu x)((\nu w, y')(y\langle w, y'\rangle | P_1 | P_2) | Q) \equiv (\nu w, y')(y\langle w, y'\rangle | P_1 | (\nu x)(P_2 | Q))
  \]
- Conversions in 2 remain obs. equivalences.

Global progress and Preservation hold as in previous interpretation.
Embedding Intuitionistic Logic in Linear Logic

- Two embeddings of intuitionistic logic in linear logic with ! [Girard]
- Curry-Howard: Intuitionistic Logic $\simeq \lambda$-calculus.
- Curry-Howard: Intuitionistic Linear Logic $\simeq$ linear $\lambda$-calculus.
- Embed $\lambda$-calculus in linear $\lambda$-calculus
- Induces CBV or CBN operational semantics [Maraist, et al. 95]
- Evaluation strategies in the scope of Curry-Howard.

Concurrent Evaluation through Curry-Howard

Can we use our interpretation to place concurrent evaluation in the scope of Curry-Howard?
Girard’s Embeddings

- Translating $T \rightarrow S$ as $!T \rightarrow S$ (corresponds to CBN)
- “Double negation” translation, using $!$ (corresponds to CBV)

Our Approach [Toninho, et al.12]

- From $\lambda$ to linear $\lambda$-calculus
- From linear $\lambda$-calculus to sequent calculus (processes)
- Compose the steps.
**Concurrent Evaluation**

**Embeddings - \( \lambda \) to linear \( \lambda \)**

### Translating \( T \to S \) as \( !T \to S \)

\[
\begin{align*}
[T \to S] & \triangleq (![T]) \to [S] \\
[b] & \triangleq b \\
[x] & \triangleq u_x \\
[\lambda x : T. M] & \triangleq \hat{x} : ![T]. \text{let } !u_x = x \text{ in } [M] \\
[M N] & \triangleq [M] ([!N])
\end{align*}
\]

### “Double negation” translation

\[
\begin{align*}
(T)^* & \triangleq !T^+ \\
(T \to S)^+ & \triangleq T^* \to S^+ \\
(b)^+ & \triangleq b \\
(x)^* & \triangleq !u_x \\
(\lambda x : T. M)^* & \triangleq !(\hat{x} : !T^+. \text{let } !u_x = x \text{ in } M^*) \\
(M N)^* & \triangleq (\text{let } !u = M^* \text{ in } u) N^*
\end{align*}
\]
Concurrent Evaluation
Embeddings - Linear $\lambda$ to processes

**Natural Deduction to Sequent Calculus**

Intuitionistic linear natural deduction can be canonically translated to linear sequents:

- If $\Gamma; \Delta \vdash M : A$ then $\Gamma; \Delta \vdash [M]_z :: z : A$
Concurrent Evaluation
Embeddings - Composing the steps

Composing $[\cdot]$ and $[[ \cdot ]_z$

$[x]_z \triangleq (\nu x)u_x \langle x \rangle. [x \leftrightarrow z]$

$[\lambda x. M]_z \triangleq z(x). (\nu y)([x \leftrightarrow y] \mid [M]_z\{y/u_x\})$

$[M N]_z \triangleq (\nu w)([M]_w \mid (\nu y)w \langle y \rangle.((!y(x).[N]_x) \mid [w \leftrightarrow z]))$

What happens in a $\beta$-redex? Copying reduction

$[(\lambda x. M) N]_z = (\nu w)(w(x). (\nu y')([x \leftrightarrow y'] \mid [M]_w\{y'/u_x\}))$

$\mid (\nu y)w \langle y \rangle.((!y(x).[N]_x) \mid [w \leftrightarrow z]))$

$\rightarrow^3 (\nu y)([M]_z\{y/u_x\} \mid !y(x).[N]_x)$
Concurrent Evaluation
Embeddings - Composing the steps

Composing $(\cdot)^*$ and $\llbracket \cdot \rrbracket_z$

$$
\llbracket x \rrbracket_z^* \triangleq \! z(a). (\nu x) u_x \langle x \rangle. [x \leftrightarrow a]
$$
$$
\llbracket \lambda x. M \rrbracket_z^* \triangleq \! z(a). a(x). (\nu y)([x \leftrightarrow y] | \llbracket M \rrbracket^*_a\{y/u_x\})
$$
$$
\llbracket M N \rrbracket_z^* \triangleq (\nu w)((\nu x)(\llbracket M \rrbracket_x^* | (\nu v)x\langle v \rangle.[v \leftrightarrow w]) | (\nu y)w\langle y \rangle.(\llbracket N \rrbracket_y^* | [w \leftrightarrow z]))
$$

What happens in a $\beta$-redex? Sharing reduction

$$
\llbracket (\lambda x. M) N \rrbracket_z^* = (\nu w)((\nu x)!(x(a).a(b).(\nu y')([b \leftrightarrow y'] | \llbracket M \rrbracket^*_a\{y'/u_x\})
| (\nu v)x\langle v \rangle.[v \leftrightarrow w]) | (\nu y)w\langle y \rangle.(\llbracket N \rrbracket_y^* | [w \leftrightarrow z]))
$$
$$
\rightarrow^5 (\nu y)(\llbracket M \rrbracket_z^*\{y/u_x\} | \llbracket N \rrbracket_y^*)
$$
Concurrent Evaluation
Summary

Strategies

- **Copying Reduction**
  - Arguments are not evaluated right away.
  - A fresh copy is evaluated per variable occurrence.
  - Reminiscent of Milner’s CBN $\pi$-calculus translation.

- **Sharing Reduction**
  - Functions evaluate in parallel with arguments.
  - Arguments are only evaluated *once* – shared.
  - CBValue on one end of the spectrum, CBNeed in the other.
  - Logical interpretation of futures!
Richer Type Theories

Motivation

Session Types
- Only express simple communication patterns.
- No interesting properties of exchanged data.
- No sophisticated properties of processes.

Answers from Logic
- Enrich the logic/types: Quantifiers, Modalities
- Dependent Type Theories
  - Integrate interpretation in a type theory
  - Reasoning about processes internally in the theory
  - Lots of challenges to overcome still.
Richer Type Theories
Where are we?

Dependent Session Types [Toninho et al.11, Pfenning et al.11]

- Two new types: $\forall x: \tau. A$ and $\exists x: \tau$
- Parametric in the language of types $\tau$.
- $\forall x: \tau. A$ - Input a term $M : \tau$, continue as $A(M)$.
- $\exists x: \tau. A$ - Output a term $M : \tau$, continue as $A(M)$.
- If $\tau$s are dependent: proof communication.
- With affirmation and proof irrelevance: proof certificates.

A Simple Example

\[
\text{indexer}_1 \triangleq !(\forall f: \text{file.pdf}(f) \rightarrow \exists g: \text{file.pdf}(g) \otimes 1)
\]
\[
\text{indexer}_2 \triangleq !(\forall f: \text{file.pdf}(f) \rightarrow \exists g: \text{file.pdf}(g) \otimes \text{agree}(f, g) \otimes 1)
\]
Richer Type Theories
Where are we?

Polymorphism and Parametricity [Pérez et al.11, Wadler11]

- Second-order quantification.
- Communication of session types / abstract protocols.
- Parametricity results in the style of System F.

Monadic Integration [Toninho et al.12]

- A λ-calculus with a linear contextual monad.
- \( \{\Gamma; \Delta \vdash z:A\} \), type of an open process expression of type \( z:A \).
- To use \( \{\Gamma; \Delta \vdash z:A\} \), provide it with suitable channels \( \Gamma \) and \( \Delta \).
- \( \text{bind}(M, \bar{x}, z.Q) \) is a process expression:
  - Evaluate \( M \) down to a monadic value, e.g. \( \{x : B \vdash P :: z:A\} \).
  - Channel list \( \bar{x} \) must satisfy \( M \)'s dependencies.
  - Run underlying process in parallel with \( Q \).
- Processes can communicate monadic values.
Conclusion

Summary
- Explored a logical interpretation of session-based concurrency
- Explain concurrency theoretic concepts using logic
- Map logical phenomena to concurrency theory
- Clean and elegant reasoning through logic.

Future work
- Fortunately, still much to do!
- A fully dependent type theory?
- Understanding definitional equality
- Inductive and Co-inductive types
- ...
Thank you!
Questions?
Linear Logic: A Logical Foundation for Concurrent Computation

Bernardo Toninho
[joint work with Luís Caires, Frank Pfenning, Jorge Pérez and Henry DeYoung]

Computer Science Department
Carnegie Mellon University
Universidade Nova de Lisboa

ENS, Lyon
Oct. 11, 2012