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

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 π-calculus [BellinScott 94]
- Research shifted to more geometric approaches.
- No real "Curry-Howard interpretation".

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 $\simeq$ λ-calculus
- Concurrency: Linear Logic $\simeq$???

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 π-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

$$\overbrace{u_1{:}A_1, \ldots, u_m{:}A_m}^{\Gamma} ; \overbrace{x_1{:}A_1, \ldots, x_n{:}A_n}^{\Delta} \Longrightarrow P :: x{:}A$$

Process P provides A along x if composed with sessions in Δ and Γ.

Cut as Composition

$$\frac{\Gamma; \Delta \Longrightarrow P :: x{:}A \quad \Gamma; \Delta', x{:}A \Longrightarrow Q :: z{:}C}{\Gamma; \Delta, \Delta' \Longrightarrow (\nu x)(P \mid Q) :: z{:}C}\ \text{cut}$$

Parallel composition of P, offering $x{:}A$ and Q, using $x{:}A$.

Identity as Renaming

$$\frac{}{\Gamma; x{:}A \Longrightarrow [x \leftrightarrow z] :: z{:}A}$$

Multiplicative Conjunction

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

$$\frac{\Gamma; \Delta, y : A, x : B \Longrightarrow Q :: z{:}C}{\Gamma; \Delta, x{:}A \otimes B \Longrightarrow x(y).Q :: z{:}C} \otimes L$$

Proof Reduction

$$\Gamma; \Delta, \Delta' \Longrightarrow (\nu x)((\nu y)x\langle y\rangle.(P_1 \mid P_2) \mid x(y).Q) :: z{:}C$$
$$\longrightarrow \quad \Gamma; \Delta, \Delta' \Longrightarrow (\nu x)(\nu y)(P_1 \mid P_2 \mid Q) :: z{:}C$$

Linear Implication

$$\frac{\Gamma; \Delta, y : A \Longrightarrow P :: x{:}B}{\Gamma; \Delta \Longrightarrow x(y).P :: x{:}A \multimap B} \multimap R$$

$$\frac{\Gamma; \Delta \Longrightarrow Q_1 :: y{:}A \quad \Gamma; \Delta', x{:}B \Longrightarrow Q_2 :: z{:}C}{\Gamma; \Delta, \Delta', x{:}A \multimap B \Longrightarrow (\nu y)x\langle y\rangle.(Q_1 \mid Q_2) :: z{:}C} \multimap L$$

Linear Implication as input. Reduction is the same as for $\otimes$.

Multiplicative Unit

$$\frac{}{\Gamma; \cdot \Longrightarrow \mathbf{0} :: x{:}\mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma; \Delta \Longrightarrow Q :: z{:}C}{\Gamma; \Delta, x{:}\mathbf{1} \Longrightarrow Q :: z{:}C}\ \mathbf{1}L$$

Proof Reduction

$$\Gamma; \Delta \Longrightarrow (\nu x)(\mathbf{0} \mid Q) :: z{:}C$$
$$\equiv \quad \Gamma; \Delta \Longrightarrow Q :: z{:}C$$

Multiplicative Unit as Termination

Additive Conjunction

$$\frac{\Gamma; \Delta \Longrightarrow P_1 :: x{:}A \quad \Gamma; \Delta \Longrightarrow P_2 :: x{:}B}{\Gamma; \Delta \Longrightarrow x.\mathsf{case}(P_1, P_2) :: x{:}A \,\&\, B} \;\&R$$

$$\frac{\Gamma; \Delta, x{:}A \Longrightarrow Q :: z{:}C}{\Gamma; \Delta, x{:}A \,\&\, B \Longrightarrow x.\mathsf{inl}; Q :: z{:}C} \;\&L_1$$

Proof Reduction

$$\Gamma; \Delta, \Delta' \Longrightarrow (\nu x)(x.\mathsf{case}(P_1, P_2) \mid x.\mathsf{inl}; Q) :: z{:}C$$
$$\longrightarrow \Gamma; \Delta, \Delta' \Longrightarrow (\nu x)(P_1 \mid Q) :: z{:}C$$

Additive Disjunction

$$\frac{\Gamma; \Delta \Longrightarrow P :: x{:}A}{\Gamma; \Delta \Longrightarrow x.\mathsf{inl}; P :: x{:}A \oplus B} \oplus R_1$$

$$\frac{\Gamma; \Delta, x{:}A \Longrightarrow Q_1 :: z{:}C \quad \Gamma; \Delta, x{:}B \Longrightarrow Q_2 :: z{:}C}{\Gamma; \Delta, x{:}A \oplus B \Longrightarrow x.\mathsf{case}(Q_1, Q_2) :: z{:}C} \oplus L$$

Same proof reductions as $\&$.

Persistent Cut

$$\frac{\Gamma; \cdot \Longrightarrow P :: x{:}A \qquad \Gamma, u{:}A; \Delta \Longrightarrow Q :: z{:}C}{\Gamma; \Delta \Longrightarrow (\nu u)(!u(x).P \mid Q) :: z{:}C}\ \mathsf{cut}^{!}$$

Parallel composition of P, offering $x{:}A$ and Q, using $u{:}A$ persistently.

Copy

$$\frac{\Gamma, u{:}A; \Delta, x{:}A \Longrightarrow P :: z{:}C}{\Gamma, u{:}A; \Delta \Longrightarrow (\nu x)u\langle x\rangle.Q :: z{:}C}\ \mathsf{copy}$$

Proof Reduction

$$\Gamma; \Delta \Longrightarrow (\nu u)(!u(x).P \mid (\nu x)u\langle x\rangle.Q) :: z{:}C$$

$$\longrightarrow \Gamma; \Delta \Longrightarrow (\nu u)(!u(x).P \mid (\nu x)(P \mid Q)) :: z{:}C$$

Exponential

$$\frac{\Gamma; \cdot \Longrightarrow P :: y{:}A}{\Gamma; \cdot \Longrightarrow !x(y).P :: x{:}!A} \; !R \qquad \frac{\Gamma, u{:}A; \Delta \Longrightarrow P :: z{:}C}{\Gamma; \Delta, x{:}!A \Longrightarrow P\{x/u\} :: z{:}C} \; !L$$

Proof reduction transforms a cut into a cut$^{!}$ (struct. equivalence).

Operational Correspondence and Subject Reduction

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

Global Progress

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

If $\vdash P :: x{:}\mathbf{1}$ and $\mathit{live}(P)$ 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

- 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. commutting inference rules).

First two correspond to reductions and structural equivalences.

Commuting Conversions induce a congruence $\simeq_c$ on typed processes

$\otimes L / \otimes L$ Commuting Conversion

$$x{:}A \otimes B, z{:}C \otimes D \Longrightarrow x(y).z(w).P \simeq_c z(w).x(y).P :: v{:}E$$

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

Typed Contextual Equivalence

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

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

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:
  - $P \sim Q :: x{:}A \multimap B$ iff $P \xrightarrow{x(y)} P'$ implies $Q \stackrel{x(y)}{\Rightarrow} Q'$ and $\forall R. \cdot \Longrightarrow R :: y{:}A$ we have $(\nu y)(P \mid R) \sim (\nu y)(Q \mid R) :: x{:}B$
  - $P \sim Q :: x{:}C$ iff $P \xrightarrow{\tau} P'$ implies $Q \Rightarrow Q'$ and $P' \sim Q' :: x{:}C$.
  - …

Proof Conversions
Issue

$\otimes L / \otimes L$ Conversion Revisited

$$(\nu x)(\nu z)(x(y).z(w).P \mid R_x \mid S_z) \sim (\nu x)(\nu z)(z(w).x(y).P \mid R_x \mid S_z) :: v{:}E?$$

- 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!

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:
  - $P \in \mathcal{L}[z{:}A \multimap B] \triangleq$ if $P \stackrel{z(y)}{\Rightarrow} P'$ 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: $\Gamma; \Delta \vdash P \mathcal{R} Q :: T$
  - If $\Gamma; \Delta, y{:}A \vdash P \mathcal{R} Q :: T$ then $\forall R. \vdash R :: y{:}A$, $\Gamma; \Delta \vdash (\nu y)(P \,|\, R) \mathcal{R} (\nu y)(Q \,|\, R) :: T$.
- Base case is inductive/coinductive on types:
  - $\vdash P \mathcal{R} Q :: x{:}A \multimap B$ iff $P \xrightarrow{x(y)} P'$ implies $Q \overset{x(y)}{\Rightarrow} Q'$ and $\forall R. \cdot \vdash R :: y{:}A$ we have $\vdash (\nu y)(P \mid R) \mathcal{R} (\nu y)(Q \mid R) :: x{:}B$
  - $\ldots$
- $\approx$ is the largest such relation.

Soundness of Commuting Conversions

If $\Gamma; \Delta \vdash P \simeq_c Q :: T$ then $\Gamma; \Delta \vdash P \approx Q :: T$

Type Isomorphism ($A \simeq B$)

Types A and B are iso. if there are proofs π_A of $B \vdash A$ and π_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 $y{:}B \vdash Q :: x{:}A$.
- $x{:}A \vdash (\nu y)(P \mid Q) \approx [x \leftrightarrow z] :: z{:}A$.
- $y{:}B \vdash (\nu x)(Q \mid P) \approx [y \leftrightarrow z] :: z{:}B$.

Validating Isomorphisms

If $A \simeq B$ then $A \simeq_S B$.

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]

Process Assignment

- Uses the same rules, but slightly different processes.
- Input clauses stay basically unchanged:
  - Since input is binding, its a synchronization point.
- Outputs are asynchronous: $x\langle y\rangle.P$ vs. $x\langle y\rangle \mid P$.

Asynchronous Assignment for $\multimap$ – Tentative

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

$$\frac{\Gamma; \Delta \vdash Q_1 :: y{:}A \quad \Gamma; \Delta', x{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta, \Delta', x{:}A \multimap B \vdash (\nu y)(x\langle y\rangle \mid Q_1 \mid Q_2) :: z{:}C} \multimap L?$$

Problem

Consider the type $A_1 \multimap (A_2 \multimap B)$:

$$\dfrac{\Delta_1 \vdash P_1 :: y_1{:}A_1 \quad \dfrac{\Delta_2 \vdash P_2 :: y_2{:}A_2 \quad \Delta_3, x{:}B \vdash Q :: z{:}C}{\Delta_2, \Delta_3, x{:}A_2 \multimap B \vdash (\nu y_2)(x\langle y_2\rangle \mid P_2 \mid Q) :: z{:}C}}{\Delta_1, \Delta_2, \Delta_3, x{:}A_1 \multimap (A_2 \multimap B) \vdash (\nu y_1)(x\langle y_1\rangle \mid P_1 \mid (\nu y_2)(x\langle y_2\rangle \mid P_2 \mid Q)) :: z{:}C}$$

- 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 Assignment for $\multimap$ – Fixed

$$\frac{\Gamma; \Delta \vdash Q_1 :: y{:}A \quad \Gamma; \Delta', x'{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta, \Delta', x{:}A \multimap B \vdash (\nu y, x')(x\langle y, x' \rangle \mid Q_1 \mid Q_2) :: z{:}C} \multimap L$$

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

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 π-calculus reduction.

Asynchronous Assignment for $\otimes$

$$\frac{\Gamma; \Delta \vdash P_1 :: y{:}A \quad \Gamma; \Delta' \vdash P_2 :: x'{:}B}{\Gamma; \Delta, \Delta' \vdash (\nu y, x')(x\langle y, x'\rangle \mid P_1 \mid P_2) :: x{:}A \otimes B} \otimes R$$

$$\frac{\Gamma; \Delta, y{:}A, x'{:}B \vdash Q :: z{:}C}{\Gamma; \Delta, x{:}A \otimes B \vdash x(y, x').Q :: z{:}C} \otimes L$$

Asynchronous Assignment for **1**

$$\frac{}{\Gamma; \cdot \vdash x\langle\rangle :: x{:}\mathbf{1}} \mathbf{1}R \qquad \frac{\Gamma; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}\mathbf{1} \vdash x().\mathbf{0} \mid Q :: z{:}C} \mathbf{1}L$$

Proof Reduction

$$\Gamma; \Delta \vdash (\nu x)(x\langle\rangle \mid x().\mathbf{0} \mid Q) :: z{:}C \longrightarrow \Gamma; \Delta \vdash Q :: z{:}C$$

Asynchronous Assignment for $\&$

$$\frac{\Gamma; \Delta \vdash P_1 :: x_1':A \quad \Gamma; \Delta \vdash P_2 :: x_2':B}{\Gamma; \Delta \vdash x.\mathsf{case}((x_1').P_1, (x_2').P_2) :: x{:}A \mathbin{\&} B} \;\&R$$

$$\frac{\Gamma; \Delta, x_1':A \vdash Q :: z{:}C}{\Gamma; \Delta, x{:}A \mathbin{\&} B \vdash (\nu x_1')(x.\mathsf{inl}\langle x_1' \rangle \mid Q) :: z{:}C} \;\&L_1$$

Proof Reduction

$$\Gamma; \Delta \vdash (\nu x)(x.\mathsf{case}((x_1').P_1, (x_2').P_2) \mid (\nu x_1')(x.\mathsf{inl}\langle x_1' \rangle \mid Q)) :: z{:}C$$

$$\longrightarrow \Gamma; \Delta \vdash (\nu x_1')(P_1 \mid Q) :: z{:}C$$

Asynchronous Assignment for !

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

$$\frac{\Gamma; \cdot \vdash P :: y{:}A}{\Gamma; \cdot \vdash (\nu u)(x\langle u\rangle \mid !u(y).P) :: x{:}!A}\ !R \qquad \frac{\Gamma, u{:}A; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta, x{:}!A \vdash x(u).Q :: z{:}C}\ !L$$

Proof Reduction

$$\Gamma; \Delta \vdash (\nu x)((\nu u)(x\langle u\rangle \mid !u(y).P) \mid x(u).Q) :: z{:}C$$
$$\longrightarrow \Gamma; \Delta \vdash (\nu u)(!u(y).P \mid Q) :: z{:}C$$

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*:

$$(\nu x)((\nu w, y')(y\langle w, y'\rangle \mid P_1 \mid P_2) \mid Q) \equiv (\nu w, y')(y\langle w, y'\rangle \mid P_1 \mid (\nu x)(P_2 \mid Q))$$

- 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 $\simeq$ λ-calculus.
- Curry-Howard: Intuitionistic Linear Logic $\simeq$ linear λ-calculus.
- Embed λ-calculus in linear λ-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 $T \to S$ as $!T \multimap S$ (corresponds to CBN)
- "Double negation" translation, using ! (corresponds to CBV)

Our Approach [Toninho, et al.12]

- From λ to linear λ-calculus
- From linear λ-calculus to sequent calculus (processes).
- Compose the steps.

Translating $T \to S$ as $!T \multimap S$

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

"Double negation" translation

$$
\begin{array}{lcl}
(T)^* & \triangleq & !T^+ \\
(T \to S)^+ & \triangleq & T^* \multimap S^* \\
(b)^+ & \triangleq & b \\
\\
(x)^* & \triangleq & !u_x \\
(\lambda x : T.\, M)^* & \triangleq & !(\hat{\lambda} x : !T^+.\, \text{let } !u_x = x \text{ in } M^*) \\
(M\, N)^* & \triangleq & (\text{let } !u = M^* \text{ in } u)\, N^*
\end{array}
$$

Natural Deduction to Sequent Calculus

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

- If $\Gamma; \Delta \vdash M : A$ then $\Gamma; \Delta \vdash [\![M]\!]_z :: z : A$

Linear λ-calculus to π-calculus translation

$$
\begin{array}{lcl}
[\![x]\!]_z & \triangleq & [x \leftrightarrow z] \\
[\![u]\!]_z & \triangleq & (\nu x)u\langle x\rangle.[x \leftrightarrow z] \\
[\![\hat{\lambda} x.M]\!]_z & \triangleq & z(x).[\![M]\!]_z \\
[\![M\,N]\!]_z & \triangleq & (\nu x)([\![M]\!]_x \mid (\nu y)x\langle y\rangle.([\![N]\!]_y \mid [x \leftrightarrow z])) \\
[\![!M]\!]_z & \triangleq & !z(x).[\![M]\!]_x \\
[\![\text{let } !u = M \text{ in } N]\!]_z & \triangleq & (\nu x)([\![M]\!]_x \mid [\![N]\!]_z\{x/u\})
\end{array}
$$

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

$$
\begin{array}{lcl}
[x]_z & \triangleq & (\nu x)u_x\langle x\rangle.[x \leftrightarrow z] \\
[\lambda x.M]_z & \triangleq & z(x).(\nu y)([x \leftrightarrow y] \mid [M]_z\{y/u_x\}) \\
[M\,N]_z & \triangleq & (\nu w)([M]_w \mid (\nu y)w\langle y\rangle.((!y(x).[N]_x) \mid [w \leftrightarrow z]))
\end{array}
$$

What happens in a β-redex? Copying reduction

$$
\begin{array}{lcl}
[(\lambda x.M)\,N]_z & = & (\nu w)(w(x).(\nu y')([x \leftrightarrow y'] \mid [M]_w\{y'/u_x\}) \\
 & & \mid (\nu y)w\langle y\rangle.((!y(x).[N]_x) \mid [w \leftrightarrow z])) \\
 & \to^3 & (\nu y)([M]_z\{y/u_x\} \mid !y(x).[N]_x)
\end{array}
$$

Composing $(\cdot)^*$ and $[\![\cdot]\!]_z$

$$
\begin{array}{lcl}
[\![x]\!]^*_z & \triangleq & !z(a).(\nu x)u_x\langle x\rangle.[x \leftrightarrow a] \\
[\![\lambda x.M]\!]^*_z & \triangleq & !z(a).a(x).(\nu y)([x \leftrightarrow y] \mid [\![M]\!]^*_a\{y/u_x\}) \\
[\![M\,N]\!]^*_z & \triangleq & (\nu w)((\nu x)([\![M]\!]^*_x \mid (\nu v)x\langle v\rangle.[v \leftrightarrow w]) \mid \\
 & & (\nu y)w\langle y\rangle.([\![N]\!]^*_y \mid [w \leftrightarrow z]))
\end{array}
$$

What happens in a β-redex? *Sharing* reduction

$$
\begin{array}{lcl}
[\![(\lambda x.M)\,N]\!]^*_z & = & (\nu w)((\nu x)(!x(a).a(b).(\nu y')([b \leftrightarrow y'] \mid [\![M]\!]^*_a\{y'/u_x\}) \\
 & & \mid (\nu v)x\langle v\rangle.[v \leftrightarrow w]) \mid (\nu y)w\langle y\rangle.([\![N]\!]^*_y \mid [w \leftrightarrow z])) \\
 & \to^5 & (\nu y)([\![M]\!]^*_z\{y/u_x\} \mid [\![N]\!]^*_y)
\end{array}
$$

Strategies

- Copying Reduction
  - Arguments are not evaluated right away.
  - A fresh copy is evaluated per variable occurrence.
  - Reminiscent of Milner's CBN π-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!

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.

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 τ.
- $\forall x{:}\tau.A$ - Input a term $M : \tau$, continue as $A(M)$.
- $\exists x{:}\tau.A$ - Output a term $M : \tau$, continue as $A(M)$.
- If τs are dependent: proof communication.
- With affirmation and proof irrelevance: proof certificates.

A Simple Example

$$\mathsf{indexer}_1 \triangleq\ !(\forall f{:}\mathsf{file}.\mathsf{pdf}(f) \multimap \exists g{:}\mathsf{file}.\mathsf{pdf}(g) \otimes \mathbf{1})$$

$$\mathsf{indexer}_2 \triangleq\ !(\forall f{:}\mathsf{file}.\mathsf{pdf}(f) \multimap \exists g{:}\mathsf{file}.\mathsf{pdf}(g) \otimes \mathsf{agree}(f,g) \otimes \mathbf{1})$$

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 Γ and Δ.
- bind$(M, \overline{x}, z.Q)$ is a process expression:
  - Evaluate M down to a monadic value, e.g. $\{x : B \vdash P :: z{:}A\}$.
  - Channel list $\overline{x}$ 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