A Logical Foundation for Session-based Concurrent Computation

Bernardo Toninho
[Co-advised by: Luís Caires and Frank Pfenning]

Computer Science Department
Carnegie Mellon University
Universidade Nova de Lisboa

Thesis Proposal

Motivation

Concurrency and Distribution

Concurrent and Distributed Systems

- Ubiquitous: From smart phone apps to business infrastructures.
- Building these systems is hard.
- Building these systems *correctly* is harder.
- *Reasoning* about these systems is even harder.

Concurrent and Distributed Systems

- Ubiquitous: From smart phone apps to business infrastructures.
- Building these systems is hard.
- Building these systems *correctly* is harder.
- *Reasoning* about these systems is even harder.

Why is it hard?

- New problems arise: deadlocks, livelocks, security, etc.
- Compositional reasoning is generally hard to do in this setting (if possible).
- Testing isn't a reasonable approach (conditions hard to replicate).

Correctness - What does it mean?

- System doesn't "get stuck" / deadlock.
- System is responsive (i.e. it exhibits some observable behavior).
- System behavior adheres to some specified protocol.

Motivation

How to approach these systems?

Correctness - What does it mean?

- System doesn't "get stuck" / deadlock.
- System is responsive (i.e. it exhibits some observable behavior).
- System behavior adheres to some specified protocol.

How do we reason about these systems?

- Language-based models (e.g. Process Calculi)
- Logics (e.g. HML, Separation Logic)
- Type Systems (e.g. I/O Types, Session Types)

Process Calculi

- Algebraic model of concurrent message-passing processes.
- Enable study of behavior and interaction of systems.
- General model of computation (e.g. π-calculus).
- Enriched with types to enforce properties on processes.

Process Calculi

- Algebraic model of concurrent message-passing processes.
- Enable study of behavior and interaction of systems.
- General model of computation (e.g. π-calculus).
- Enriched with types to enforce properties on processes.

Types and the π-calculus

- Specify what data can be sent along channels (Simple types).
- Specify input/output capabilities of channels (I/O types).
- Specify communication behavior along channels (Session types).

Issues

- Many different, sometimes *ad-hoc*, language features.
- Hard to reason about these languages in a uniform way.
- Challenging concerns when compared to non-concurrent setting:
  - Interactive behavior.
  - Non-local state and irreversible actions.
  - Resource awareness
- No deep connection with logic.

Issues

- Many different, sometimes *ad-hoc*, language features.
- Hard to reason about these languages in a uniform way.
- Challenging concerns when compared to non-concurrent setting:
  - Interactive behavior.
  - Non-local state and irreversible actions.
  - Resource awareness
- No deep connection with logic.

Why does logic matter?

- New means of reasoning about concurrent phenomena.
- Good metalogical properties map to good program properties.
- Enables compositional and incremental study of language features.

Thesis Statement

Thesis Statement

Linear logic, specifically in its intuitionistic formulation, is a suitable logical foundation for message-passing concurrent computation, providing an elegant framework in which to express and reason about a multitude of naturally occurring phenomena in such a concurrent setting.

Outline

- Linear logic and message-passing concurrency:
  - Background
  - Basic interpretation
  - Properties
  - Extensions

Outline

- Linear logic and message-passing concurrency:
  - Background
  - Basic interpretation
  - Properties
  - Extensions
- A concurrent programming language:
  - Monadic encapsulation of concurrent computation
  - Recursion
  - Reconciliation with Logic

Outline

- Linear logic and message-passing concurrency:
  - Background
  - Basic interpretation
  - Properties
  - Extensions
- A concurrent programming language:
  - Monadic encapsulation of concurrent computation
  - Recursion
  - Reconciliation with Logic
- Reasoning Techniques:
  - Linear logical relations
  - Parametricity
  - Dependent Types

Linear Logic [Girard 87]

- A logic of resources and interaction.
- Resource independence captures parallelism.
- Linear logic as the logic of concurrency?

Linear Logic [Girard 87]

- A logic of resources and interaction.
- Resource independence captures parallelism.
- 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]
- No real "Curry-Howard interpretation"

Session Types [Honda 93]

- A structuring discipline for message passing concurrency.
- Session: Predetermined sequence of interactions along a channel.
- Session Types: Types are descriptions of communication behavior.
- Type correct programs adhere to the session discipline.

Session Types [Honda 93]

- A structuring discipline for message passing concurrency.
- Session: Predetermined sequence of interactions along a channel.
- Session Types: Types are descriptions of communication behavior.
- Type correct programs adhere to the session discipline.

Session Types and ILL (SILL) [CairesPfenning10]

- Its possible to interpret session types as linear logic propositions.
- Proofs as typing derivations.
- Proof dynamics as process dynamics.

Deviating from SILL

- Explicit linear forwarders.
- Syntax-driven assignment.
- Syntax as a basis for a concurrent, session-typed language.
- Consistently mapped to π-calculus processes.

Deviating from SILL

- Explicit linear forwarders.
- Syntax-driven assignment.
- Syntax as a basis for a concurrent, session-typed language.
- Consistently mapped to π-calculus processes.

Propositions as Types

$A \otimes B$	Output a session of type A and continue as B
$A \multimap B$	Input a session of type A and continue as B
$A \mathbin{\&} B$	Offer a choice between a session of type A or B
$A \oplus B$	Select a session of type A or B
$!A$	A persistent session of type A
$\mathbf{1}$	Terminated session

Typing Judgment

$$\overbrace{A_1, \ldots, \quad A_m}^{\Gamma}; \overbrace{A_1, \ldots, \quad A_n}^{\Delta} \vdash \quad A$$

Typing Judgment

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

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

Typing Judgment

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

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

Cut as Composition

$$\frac{\Gamma; \Delta \vdash \qquad A \quad \Gamma; \Delta', \quad A \vdash \qquad C}{\Gamma; \Delta, \Delta' \vdash \qquad C}\text{cut}$$

Typing Judgment

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

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

Cut as Composition

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', \quad A \vdash \qquad C}{\Gamma; \Delta, \Delta' \vdash \qquad C}\ \text{cut}$$

Typing Judgment

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

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

Cut as Composition

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma; \Delta, \Delta' \vdash \qquad\qquad\qquad\qquad C}\ \mathsf{cut}$$

Typing Judgment

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

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

Cut as Composition

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma; \Delta, \Delta' \vdash \mathsf{new}\ x.(P \mid Q) :: z{:}C}\ \text{cut}$$

Typing Judgment

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

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

Cut as Composition

$$\frac{\Gamma;\Delta \vdash P :: x{:}A \quad \Gamma;\Delta',x{:}A \vdash Q :: z{:}C}{\Gamma;\Delta,\Delta' \vdash \mathsf{new}\ x.(P \mid Q) :: z{:}C}\ \mathsf{cut}$$

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

Identity as Forwarding

$$\frac{}{\Gamma;\quad A \vdash \qquad\qquad A}\ \mathsf{id}$$

Typing Judgment

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

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

Cut as Composition

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma; \Delta, \Delta' \vdash \mathsf{new}\ x.(P \mid Q) :: z{:}C}\ \text{cut}$$

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

Identity as Forwarding

$$\frac{}{\Gamma; x{:}A \vdash \qquad\qquad\qquad A}\ \text{id}$$

Typing Judgment

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

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

Cut as Composition

$$\frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma; \Delta, \Delta' \vdash \mathsf{new}\ x.(P \mid Q) :: z{:}C}\ \mathsf{cut}$$

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

Identity as Forwarding

$$\frac{}{\Gamma; x{:}A \vdash \mathsf{fwd}\ x\ z :: z{:}A}\ \mathsf{id}$$

Session Output

$$\frac{\Gamma; \Delta \vdash \qquad A \quad \Gamma; \Delta' \vdash \qquad B}{\Gamma; \Delta, \Delta' \vdash \qquad A \otimes B} \otimes R$$

Session Output

$$\frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta' \vdash Q :: z{:}B}{\Gamma; \Delta, \Delta' \vdash \qquad\qquad A \otimes B}\ \otimes R$$

Session Output

$$\frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta' \vdash Q :: z{:}B}{\Gamma; \Delta, \Delta' \vdash \mathsf{output}\ z\ (y.P); Q :: z{:}A \otimes B} \otimes R$$

Session Output

$$\frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta' \vdash Q :: z{:}B}{\Gamma; \Delta, \Delta' \vdash \mathsf{output}\ z\ (y.P); Q :: z{:}A \otimes B}\ \otimes R$$

$$\frac{\Gamma; \Delta, \quad A, \quad B \vdash \qquad C}{\Gamma; \Delta, \quad A \otimes B \vdash \qquad C}\ \otimes L$$

Session Output

$$\frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta' \vdash Q :: z{:}B}{\Gamma; \Delta, \Delta' \vdash \text{output } z\ (y.P); Q :: z{:}A \otimes B} \otimes R$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash R :: z{:}C}{\Gamma; \Delta, \quad A \otimes B \vdash \qquad C} \otimes L$$

Session Output

$$\frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta' \vdash Q :: z{:}B}{\Gamma; \Delta, \Delta' \vdash \mathsf{output}\ z\ (y.P); Q :: z{:}A \otimes B} \otimes R$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash R :: z{:}C}{\Gamma; \Delta, x{:}A \otimes B \vdash y \leftarrow \mathsf{input}\ x; R :: z{:}C} \otimes L$$

Session Output

$$\frac{\Gamma; \Delta \vdash P :: y{:}A \quad \Gamma; \Delta' \vdash Q :: z{:}B}{\Gamma; \Delta, \Delta' \vdash \text{output } z\ (y.P); Q :: z{:}A \otimes B} \otimes R$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash R :: z{:}C}{\Gamma; \Delta, x{:}A \otimes B \vdash y \leftarrow \text{input } x; R :: z{:}C} \otimes L$$

Proof Reduction

$$\Gamma; \Delta, \Delta' \vdash \text{new } x.((\text{output } x\ (y.P); Q) \mid y \leftarrow \text{input } x; R) :: z{:}C$$

Session Output

$$\frac{\Gamma;\Delta \vdash P :: y{:}A \quad \Gamma;\Delta' \vdash Q :: z{:}B}{\Gamma;\Delta,\Delta' \vdash \mathsf{output}\ z\ (y.P);Q :: z{:}A \otimes B} \otimes R$$

$$\frac{\Gamma;\Delta, y : A, x : B \vdash R :: z{:}C}{\Gamma;\Delta, x{:}A \otimes B \vdash y \leftarrow \mathsf{input}\ x; R :: z{:}C} \otimes L$$

Proof Reduction

$$\Gamma;\Delta,\Delta' \vdash \mathsf{new}\ x.((\mathsf{output}\ x\ (y.P);Q) \mid y \leftarrow \mathsf{input}\ x; R) :: z{:}C$$
$$\longrightarrow \quad \Gamma;\Delta,\Delta' \vdash \mathsf{new}\ x.(Q \mid \mathsf{new}\ y.(P \mid R)) :: z{:}C$$

Session Input

$$\frac{\Gamma; \Delta, \quad A \vdash \qquad B}{\Gamma; \Delta \vdash \qquad A \multimap B} \multimap R$$

Session Input

$$\frac{\Gamma; \Delta, x : A \vdash P :: z{:}B}{\Gamma; \Delta \vdash \qquad\qquad\qquad A \multimap B} \multimap R$$

Session Input

$$\frac{\Gamma; \Delta, x : A \vdash P :: z{:}B}{\Gamma; \Delta \vdash x \leftarrow \mathsf{input}\ z; P :: z{:}A \multimap B}\ {\multimap}R$$

Session Input

$$\frac{\Gamma; \Delta, x : A \vdash P :: z{:}B}{\Gamma; \Delta \vdash x \leftarrow \text{input } z; P :: z{:}A \multimap B} \multimap R$$

$$\frac{\Gamma; \Delta \vdash \qquad\qquad A \quad \Gamma; \Delta', \quad B \vdash \qquad\qquad C}{\Gamma; \Delta, \Delta', \quad A \multimap B \vdash \qquad\qquad\qquad\qquad C} \multimap L$$

Session Input

$$\frac{\Gamma; \Delta, x : A \vdash P :: z{:}B}{\Gamma; \Delta \vdash x \leftarrow \mathsf{input}\ z; P :: z{:}A \multimap B}\ {\multimap}R$$

$$\frac{\Gamma; \Delta \vdash Q_1 :: y{:}A \quad \Gamma; \Delta', x{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta, \Delta', \quad A \multimap B \vdash \qquad\qquad\qquad\qquad C}\ {\multimap}L$$

Session Input

$$\frac{\Gamma; \Delta, x : A \vdash P :: z{:}B}{\Gamma; \Delta \vdash x \leftarrow \text{input } z; P :: z{:}A \multimap B} \multimap R$$

$$\frac{\Gamma; \Delta \vdash Q_1 :: y{:}A \quad \Gamma; \Delta', x{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta, \Delta', x{:}A \multimap B \vdash \text{output } x\ (y.Q_1); Q_2) :: z{:}C} \multimap L$$

Session Input

$$\frac{\Gamma; \Delta, x : A \vdash P :: z{:}B}{\Gamma; \Delta \vdash x \leftarrow \text{input } z; P :: z{:}A \multimap B} \multimap R$$

$$\frac{\Gamma; \Delta \vdash Q_1 :: y{:}A \quad \Gamma; \Delta', x{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta, \Delta', x{:}A \multimap B \vdash \text{output } x\ (y.Q_1); Q_2) :: z{:}C} \multimap L$$

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

Multiplicative Unit

$$\frac{}{\Gamma ; \cdot \vdash \mathbf{1}}\ \mathbf{1}R$$

Multiplicative Unit

$$\frac{}{\Gamma; \cdot \vdash \textsf{close}\ z :: z{:}\mathbf{1}}\ \mathbf{1}R$$

Multiplicative Unit

$$\frac{}{\Gamma; \cdot \vdash \text{close } z :: z{:}\mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma; \Delta \vdash \qquad C}{\Gamma; \Delta, \quad \mathbf{1} \vdash \qquad C}\ \mathbf{1}L$$

Multiplicative Unit

$$\frac{}{\Gamma; \cdot \vdash \text{close } z :: z{:}\mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta, \quad \mathbf{1} \vdash \qquad\qquad C}\ \mathbf{1}L$$

Multiplicative Unit

$$\frac{}{\Gamma; \cdot \vdash \mathsf{close}\ z :: z{:}\mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta, x{:}\mathbf{1} \vdash \mathsf{wait}\ x; Q :: z{:}C}\ \mathbf{1}L$$

Proof Reduction

$$\Gamma; \Delta \vdash \mathsf{new}\ x.(\mathsf{close}\ x \mid (\mathsf{wait}\ x; Q)) :: z{:}C$$

Multiplicative Unit

$$\frac{}{\Gamma; \cdot \vdash \mathsf{close}\ z :: z{:}\mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta, x{:}\mathbf{1} \vdash \mathsf{wait}\ x; Q :: z{:}C}\ \mathbf{1}L$$

Proof Reduction

$$\Gamma; \Delta \vdash \mathsf{new}\ x.(\mathsf{close}\ x \mid (\mathsf{wait}\ x; Q)) :: z{:}C$$

$$\longrightarrow \quad \Gamma; \Delta \vdash Q :: z{:}C$$

Additive Conjunction

$$\frac{\Gamma ; \Delta \vdash \qquad\qquad A \quad \Gamma ; \Delta \vdash \qquad\qquad B}{\Gamma ; \Delta \vdash \qquad\qquad\qquad\qquad A \mathbin{\&} B} \; \& R$$

Additive Conjunction

$$\frac{\Gamma ; \Delta \vdash P_1 :: x{:}A \quad \Gamma ; \Delta \vdash P_2 :: x{:}B}{\Gamma ; \Delta \vdash \qquad\qquad\qquad\qquad A \mathbin{\&} B} \;\&R$$

Additive Conjunction

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

Additive Conjunction

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

$$\frac{\Gamma; \Delta, \quad A \vdash \qquad\qquad C}{\Gamma; \Delta, \quad A \mathbin{\&} B \vdash \qquad\qquad\qquad C} \;\& L_1$$

Additive Conjunction

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

$$\frac{\Gamma; \Delta, {\color{red}x{:}}A \vdash {\color{red}Q :: z{:}}C}{\Gamma; \Delta, \quad A \mathbin{\&} B \vdash \qquad\qquad C} \;\&L_1$$

Additive Conjunction

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

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

Additive Conjunction

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

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

Proof Reduction

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

$$\longrightarrow \Gamma; \Delta, \Delta' \vdash \mathsf{new}\ x.(P_1 \mid Q) :: z{:}C$$

Additive Conjunction

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

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

Proof Reduction

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

Additive disjunction $\oplus$ is dual.

Persistent Cut

$$\frac{\Gamma; \cdot \vdash A \qquad \Gamma, A; \Delta \vdash C}{\Gamma; \Delta \vdash C}\ \mathsf{cut}^{!}$$

Persistent Cut

$$\frac{\Gamma; \cdot \vdash P :: x{:}A \qquad \Gamma, u{:}A; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta \vdash \qquad C}\ \mathsf{cut}^!$$

Persistent Cut

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

Persistent Cut

$$\frac{\Gamma; \cdot \vdash P :: x{:}A \qquad \Gamma, u{:}A; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta \vdash \text{new } u.(x \leftarrow \text{!input } u\ P \mid Q) :: z{:}C}\ \text{cut}^!$$

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

Persistent Cut

$$\frac{\Gamma; \cdot \vdash P :: x{:}A \qquad \Gamma, u{:}A; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta \vdash \text{new } u.(x \leftarrow \text{!input } u\ P \mid Q) :: z{:}C}\ \text{cut}^!$$

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

Copy

$$\frac{\Gamma, \quad A; \Delta, \quad A \vdash \qquad\qquad C}{\Gamma, \quad A; \Delta \vdash \qquad\qquad\qquad\qquad C}\ \text{copy}$$

Persistent Cut

$$\frac{\Gamma; \cdot \vdash P :: x{:}A \qquad \Gamma, u{:}A; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta \vdash \text{new } u.(x \leftarrow \text{!input } u\ P \mid Q) :: z{:}C}\ \text{cut}^!$$

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

Copy

$$\frac{\Gamma, {\color{red}u}{:}A; \Delta, {\color{red}x}{:}A \vdash {\color{red}R :: z}{:}C}{\Gamma, \quad A; \Delta \vdash \qquad\qquad C}\ \text{copy}$$

Persistent Cut

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

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

Copy

$$\frac{\Gamma, u{:}A; \Delta, x{:}A \vdash R :: z{:}C}{\Gamma, u{:}A; \Delta \vdash x \leftarrow \mathsf{copy}\ u; R :: z{:}C}\ \mathsf{copy}$$

Persistent Cut

$$\frac{\Gamma; \cdot \vdash P :: x{:}A \qquad \Gamma, u{:}A; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta \vdash \text{new } u.(x \leftarrow \text{!input } u\ P \mid Q) :: z{:}C}\ \text{cut}^!$$

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

Copy

$$\frac{\Gamma, u{:}A; \Delta, x{:}A \vdash R :: z{:}C}{\Gamma, u{:}A; \Delta \vdash x \leftarrow \text{copy } u; R :: z{:}C}\ \text{copy}$$

Proof Reduction

$$\Gamma; \Delta \vdash \text{new } u.(x \leftarrow \text{!input } u\ P \mid (x \leftarrow \text{copy } u; Q)) :: z{:}C$$

$$\longrightarrow \Gamma; \Delta \vdash \text{new } u.(x \leftarrow \text{!input } u\ P \mid (\text{new } x.(P \mid Q))) :: z{:}C$$

Exponential

$$\frac{\Gamma ; \cdot \vdash A}{\Gamma ; \cdot \vdash\ !A}\ !R$$

Exponential

$$\frac{\Gamma; \cdot \vdash P :: x{:}A}{\Gamma; \cdot \vdash \qquad\qquad\qquad\qquad !A} \; !R$$

Exponential

$$\frac{\Gamma; \cdot \vdash P :: x{:}A}{\Gamma; \cdot \vdash \textsf{output}\ z\ !(x.P) :: z{:}!A}\ !R$$

Exponential

$$\frac{\Gamma ; \cdot \vdash P :: x{:}A}{\Gamma ; \cdot \vdash \text{output } z\ !(x.P) :: z{:}!A}\ !R \qquad \frac{\Gamma, \quad A ; \Delta \vdash \qquad C}{\Gamma ; \Delta, \quad !A \vdash \qquad C}\ !L$$

Exponential

$$\frac{\Gamma; \cdot \vdash P :: x{:}A}{\Gamma; \cdot \vdash \text{output } z\ !(x.P) :: z{:}!A}\ !R \qquad \frac{\Gamma, u{:}A; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta,\quad !A \vdash \qquad C}\ !L$$

Exponential

$$\frac{\Gamma; \cdot \vdash P :: x{:}A}{\Gamma; \cdot \vdash \text{output } z\ !(x.P) :: z{:}!A}\ !R \qquad \frac{\Gamma, u{:}A; \Delta \vdash Q :: z{:}C}{\Gamma; \Delta, x{:}!A \vdash u \leftarrow \text{input } x; Q :: z{:}C}\ !L$$

Proof reduction transforms a cut into a $\text{cut}^!$

Type Preservation

If $\Gamma; \Delta \vdash P :: z{:}A$ and $P \longrightarrow P'$ then $\Gamma; \Delta \vdash P' :: z{:}A$

Type Preservation

If $\Gamma; \Delta \vdash P :: z{:}A$ and $P \longrightarrow P'$ then $\Gamma; \Delta \vdash P' :: z{:}A$

Global Progress

If $\cdot; \cdot \vdash P :: z{:}\mathbf{1}$ then either $P = \mathsf{new}\ \overline{x}.(!Q; \mathsf{close}\ z)$ or $P \longrightarrow P'$

The logical basis gives us deadlock-freedom and session fidelity “for free”.

Limitations of Session Types

- Hard to extend beyond simple communication patterns.
- No uniform way of expressing properties of exchanged *data*.
- Can scale to richer settings, but technically quite challenging.

Limitations of Session Types

- Hard to extend beyond simple communication patterns.
- No uniform way of expressing properties of exchanged *data*.
- Can scale to richer settings, but technically quite challenging.

Logical Foundation

- New means of reasoning about concurrent phenomena.
- Compositional and incremental study of language features:
  - Value-dependent Session Types
  - Polymorphic Session Types

Value-dependent Session Types [Toninho et al.11]

- Two new types: $\forall x{:}\tau.A$ and $\exists x{:}\tau.A$
- 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.

Value-dependent Session Types [Toninho et al.11]

- Two new types: $\forall x{:}\tau.A$ and $\exists x{:}\tau.A$
- 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.

A Simple Example

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

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

Extend the Judgment

$$\Psi; \Gamma; \Delta \vdash P :: z{:}A$$

Ψ accounts for variables from the language of τ.

Extend the Judgment

$$\Psi; \Gamma; \Delta \vdash P :: z{:}A$$

Ψ accounts for variables from the language of τ.

Universal Quantification

$$\frac{\Psi, \qquad \tau; \Gamma; \Delta \vdash \qquad\qquad A}{\Psi; \Gamma; \Delta \vdash \qquad\qquad\qquad\qquad \forall x{:}\tau.A}\ \forall R$$

Extend the Judgment

$$\Psi; \Gamma; \Delta \vdash P :: z{:}A$$

Ψ accounts for variables from the language of τ.

Universal Quantification

$$\frac{\Psi, x : \tau; \Gamma; \Delta \vdash P :: z{:}A}{\Psi; \Gamma; \Delta \vdash \qquad\qquad \forall x{:}\tau.A}\ \forall R$$

Extend the Judgment

$$\Psi; \Gamma; \Delta \vdash P :: z{:}A$$

Ψ accounts for variables from the language of τ.

Universal Quantification

$$\frac{\Psi, x : \tau; \Gamma; \Delta \vdash P :: z{:}A}{\Psi; \Gamma; \Delta \vdash x \leftarrow \mathsf{input}\ z; P :: z{:}\forall x{:}\tau.A}\ \forall R$$

$$\frac{\Psi \vdash M : \tau \quad \Psi; \Gamma; \Delta, \quad A\{M/x\} \vdash \qquad C}{\Psi; \Gamma; \Delta, \quad \forall x{:}\tau.A \vdash \qquad C}\ \forall L$$

Extend the Judgment

$$\Psi; \Gamma; \Delta \vdash P :: z{:}A$$

Ψ accounts for variables from the language of τ.

Universal Quantification

$$\frac{\Psi, x : \tau; \Gamma; \Delta \vdash P :: z{:}A}{\Psi; \Gamma; \Delta \vdash x \leftarrow \mathrm{input}\ z; P :: z{:}\forall x{:}\tau.A}\ \forall R$$

$$\frac{\Psi \vdash M : \tau \quad \Psi; \Gamma; \Delta, \quad A\{M/x\} \vdash \qquad C}{\Psi; \Gamma; \Delta, \quad \forall x{:}\tau.A \vdash \qquad C}\ \forall L$$

Extend the Judgment

$$\Psi; \Gamma; \Delta \vdash P :: z{:}A$$

Ψ accounts for variables from the language of τ.

Universal Quantification

$$\frac{\Psi, x : \tau; \Gamma; \Delta \vdash P :: z{:}A}{\Psi; \Gamma; \Delta \vdash x \leftarrow \text{input } z; P :: z{:}\forall x{:}\tau.A}\ \forall R$$

$$\frac{\Psi \vdash M : \tau \quad \Psi; \Gamma; \Delta, x{:}A\{M/x\} \vdash Q :: z{:}C}{\Psi; \Gamma; \Delta, \quad \forall x{:}\tau.A \vdash \qquad C}\ \forall L$$

Extend the Judgment

$$\Psi; \Gamma; \Delta \vdash P :: z{:}A$$

Ψ accounts for variables from the language of τ.

Universal Quantification

$$\frac{\Psi, x : \tau; \Gamma; \Delta \vdash P :: z{:}A}{\Psi; \Gamma; \Delta \vdash x \leftarrow \text{input } z; P :: z{:}\forall x{:}\tau.A}\ \forall R$$

$$\frac{\Psi \vdash M : \tau \quad \Psi; \Gamma; \Delta, x{:}A\{M/x\} \vdash Q :: z{:}C}{\Psi; \Gamma; \Delta, x{:}\forall x{:}\tau.A \vdash \text{output } x\ M; Q :: z{:}C}\ \forall L$$

Refining Quantification [Toninho et al.11, Pfenning et al.11]

- Proof Irrelevance: $[\tau]$ – Proofs of τ are not used at runtime.
- Affirmation: $\Diamond_K \tau$ – K affirms the existence of a proof of τ.

Refining Quantification [Toninho et al.11, Pfenning et al.11]

- Proof Irrelevance: $[\tau]$ – Proofs of τ are not used at runtime.
- Affirmation: $\Diamond_K \tau$ – K affirms the existence of a proof of τ.

Refining the Example

$$\mathrm{Indexer}_{\Diamond} \triangleq \forall f{:}\mathrm{file}.\forall p{:}[\mathrm{pdf}(f)].\exists g{:}\mathrm{file}.\exists q_1{:}[\mathrm{pdf}(g)].\exists q_2{:}\Diamond_I[\mathrm{agree}(f,g)] \otimes \mathbf{1}$$

Refining Quantification [Toninho et al.11, Pfenning et al.11]

- Proof Irrelevance: $[\tau]$ – Proofs of τ are not used at runtime.
- Affirmation: $\Diamond_K \tau$ – K affirms the existence of a proof of τ.

Refining the Example

$$\mathrm{Indexer}_\Diamond \triangleq \forall f{:}\mathrm{file}.\forall p{:}[\mathrm{pdf}(f)].\exists g{:}\mathrm{file}.\exists q_1{:}[\mathrm{pdf}(g)].\exists q_2{:}\Diamond_I[\mathrm{agree}(f,g)] \otimes \mathbf{1}$$

$\Diamond_I[\mathrm{agree}(f,g)]$ is a signed certificate of the agreement of f and g.

Parametric Polymorphism [Pérez et al.11, Wadler11]

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

Universal Quantification (Second Order)

$$\frac{\Omega, X; \Gamma; \Delta \vdash \qquad A}{\Omega; \Gamma; \Delta \vdash \qquad \forall X.A}\ \forall R2$$

Parametric Polymorphism [Pérez et al.11, Wadler11]

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

Universal Quantification (Second Order)

$$\frac{\Omega, X; \Gamma; \Delta \vdash P :: z{:}A}{\Omega; \Gamma; \Delta \vdash \qquad\qquad\qquad\qquad \forall X.A}\ \forall R2$$

Parametric Polymorphism [Pérez et al.11, Wadler11]

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

Universal Quantification (Second Order)

$$\frac{\Omega, X; \Gamma; \Delta \vdash P :: z{:}A}{\Omega; \Gamma; \Delta \vdash X \leftarrow \mathsf{input}\ z; P :: z{:}\forall X.A}\ \forall R2$$

Parametric Polymorphism [Pérez et al.11, Wadler11]

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

Universal Quantification (Second Order)

$$\frac{\Omega, X; \Gamma; \Delta \vdash P :: z{:}A}{\Omega; \Gamma; \Delta \vdash X \leftarrow \mathsf{input}\ z; P :: z{:}\forall X.A}\ \forall R2$$

$$\frac{\Omega \vdash B\ \mathsf{type} \quad \Omega; \Gamma; \Delta, \quad A\{B/X\} \vdash \qquad C}{\Omega; \Gamma; \Delta, \quad \forall X.A \vdash \qquad C}\ \forall L2$$

Parametric Polymorphism [Pérez et al.11, Wadler11]

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

Universal Quantification (Second Order)

$$\frac{\Omega, X; \Gamma; \Delta \vdash P :: z{:}A}{\Omega; \Gamma; \Delta \vdash X \leftarrow \mathsf{input}\ z; P :: z{:}\forall X.A}\ \forall R2$$

$$\frac{\Omega \vdash B\ \mathsf{type} \quad \Omega; \Gamma; \Delta, {\color{red}x}{:}A\{B/X\} \vdash {\color{red}Q :: z}{:}C}{\Omega; \Gamma; \Delta, \quad \forall X.A \vdash \qquad\qquad C}\ \forall L2$$

Parametric Polymorphism [Pérez et al.11, Wadler11]

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

Universal Quantification (Second Order)

$$\frac{\Omega, X; \Gamma; \Delta \vdash P :: z{:}A}{\Omega; \Gamma; \Delta \vdash X \leftarrow \mathsf{input}\ z; P :: z{:}\forall X.A}\ \forall R2$$

$$\frac{\Omega \vdash B\ \mathsf{type} \quad \Omega; \Gamma; \Delta, x{:}A\{B/X\} \vdash Q :: z{:}C}{\Omega; \Gamma; \Delta, x{:}\forall X.A \vdash \mathsf{output}\ x\ B; Q :: z{:}C}\ \forall L2$$

Parametric Polymorphism

$$\mathsf{CloudServer} \triangleq \forall X.!(\mathsf{api} \multimap X) \multimap !X$$

Parametric Polymorphism

$$\mathsf{CloudServer} \triangleq \forall X.!(\mathsf{api} \multimap X) \multimap !X$$

Specifies a generic service that:

- Inputs any type (e.g. GMaps).

Parametric Polymorphism

$$\mathsf{CloudServer} \triangleq \forall X.!(\mathsf{api} \multimap X) \multimap !X$$

Specifies a generic service that:

- Inputs any type (e.g. GMaps).
- Inputs a persistent session that, given an implementation of api, provides the previously input type (e.g. $!(\mathsf{api} \multimap \mathsf{GMaps})$).

Parametric Polymorphism

$$\mathsf{CloudServer} \triangleq \forall X.!(\mathsf{api} \multimap X) \multimap !X$$

Specifies a generic service that:

- Inputs any type (e.g. GMaps).
- Inputs a persistent session that, given an implementation of api, provides the previously input type (e.g. $!(\mathsf{api} \multimap \mathsf{GMaps})$).
- Composes the received session accordingly, subsequently providing a persistent session of the given type (e.g. !GMaps).

Other concurrency-theoretic features that can be understood logically (and vice-versa):

- Asynchronous communication [DeYoung et al. 12]
- Concurrent evaluation strategies for λ-calculus [Toninho et al. 12].
- Session type isomorphisms [Pérez et al. 12].

A Concurrent Programming Language
Overview

Programming with Processes

- Syntax focuses solely on communication behavior.
- No “values” nor functions.
- Hard to write interesting programs.

A Concurrent Programming Language

Overview

Programming with Processes

- Syntax focuses solely on communication behavior.
- No “values” nor functions.
- Hard to write interesting programs.

Towards a Concurrent Programming Language

- Approximate the syntax of the logical interpretation.
- Introduce recursion for expressiveness.
- Combine concurrent session-typed processes with λ-calculus.

Programming with Processes

- Syntax focuses solely on communication behavior.
- No "values" nor functions.
- Hard to write interesting programs.

Towards a Concurrent Programming Language

- Approximate the syntax of the logical interpretation.
- Introduce recursion for expressiveness.
- Combine concurrent session-typed processes with λ-calculus.

Types

- $\tau \supset A$ and $\tau \wedge A$ – Input and output of data.
- $\{\overline{a_i{:}A_i} \vdash c{:}A\}$ – Encapsulation of open process expressions.

Contextual Monad

- Isolate concurrency (and linearity) in a contextual monad.
- Embed into a typical λ-calculus with recursive types.
- Functional programs can refer to processes and vice-versa.

Contextual Monad

- Isolate concurrency (and linearity) in a contextual monad.
- Embed into a typical λ-calculus with recursive types.
- Functional programs can refer to processes and vice-versa.

Typing Rules

$$\frac{\Delta = \mathsf{lin}(a_i{:}A_i) \quad \Gamma = \mathsf{shd}(a_i{:}A_i) \quad \Psi;\Gamma;\Delta \vdash P :: c{:}A}{\Psi \Vdash c \leftarrow \{P_{c,\overline{a_i}}\} \leftarrow \overline{a_i{:}A_i} : \{\overline{a_i{:}A_i} \vdash c{:}A\}}\ \{\,\}I$$

Contextual Monad

- Isolate concurrency (and linearity) in a contextual monad.
- Embed into a typical λ-calculus with recursive types.
- Functional programs can refer to processes and vice-versa.

Typing Rules

$$\frac{\Delta = \mathrm{lin}(a_i{:}A_i) \quad \Gamma = \mathrm{shd}(a_i{:}A_i) \quad \Psi; \Gamma; \Delta \vdash P :: c{:}A}{\Psi \Vdash c \leftarrow \{P_{c,\overline{a_i}}\} \leftarrow \overline{a_i{:}A_i} : \{\overline{a_i{:}A_i} \vdash c{:}A\}}\ \{\,\}I$$

$$\frac{\Delta = \mathrm{lin}(\overline{a_i{:}A_i}) \quad \Gamma \supseteq \mathrm{shd}(\overline{a_i{:}A_i}) \quad \Psi \Vdash \color{red}{M} : \{\overline{a_i{:}A_i} \vdash c{:}A\} \quad \Psi; \Gamma; \Delta', c{:}A \vdash \color{red}{Q_c} :: d{:}D}{\Psi; \Gamma; \Delta, \Delta' \vdash \color{red}{c \leftarrow M \leftarrow \overline{a_i}; Q_c} :: d{:}D}\ \{\,\}E$$

Stream Session

- Output of an infinite sequence of naturals, counting up.
- A coinductive session (as a recursive type):

```
stype natStream = nat /\ intStream
```

Stream Session

- Output of an infinite sequence of naturals, counting up.
- A coinductive session (as a recursive type):

```
stype natStream = nat /\ intStream
```

Stream Transducer

To write a recursive process, we write a recursive function:

```
trans : (nat -> nat) ->
              {d:natStream |- c:natStream}
c <- trans f =
{ n1 <- input d
  output c (f n1)
  c′ <- trans f <- d
  fwd c c′ }
```

Divergence

- Recursive types introduce divergence.
- Logical soundness is therefore lost.

Divergence

- Recursive types introduce divergence.
- Logical soundness is therefore lost.

Recovering Non-Divergence [Toninho et al. 14]

- Restrict to inductive and coinductive types.
- Only allow productive definitions:
  - Process definitions must be *guarded*.
  - Self-interaction with recursive calls must be disallowed.

Divergence

- Recursive types introduce divergence.
- Logical soundness is therefore lost.

Recovering Non-Divergence [Toninho et al. 14]

- Restrict to inductive and coinductive types.
- Only allow productive definitions:
  - Process definitions must be *guarded*.
  - Self-interaction with recursive calls must be disallowed.

```
c <- P = {output c 0 ; c′ <- P ; x <- input c′; fwd c c′}
```

Loops after first output...

Reasoning using Logic

- Up to this point we have identified concurrent "features" that can be justified logically.
- Can we do more? Can we use the logical foundation to *reason* about concurrent programs?

Reasoning using Logic

- Up to this point we have identified concurrent “features” that can be justified logically.
- Can we do more? Can we use the logical foundation to *reason* about concurrent programs?

Logical Relations

- Standard PL technique for λ-calculus.
- Mostly unexplored for typed process calculi.
- Useful for showing non-trivial properties (e.g. termination, confluence, etc.).

Linear Logical Relations [Pérez et al. 12, Caires et al. 13, Toninho et al. 14]

- Uniform framework for our session-typed processes.
- Scales to handle polymorphism and coinductive types.

Linear Logical Relations [Pérez et al. 12, Caires et al. 13, Toninho et al. 14]

- Uniform framework for our session-typed processes.
- Scales to handle polymorphism and coinductive types.

Informal Idea - Unary Case

- Candidates – Set of processes satisfying the relevant property.
- Inductive on typing: $\mathcal{L}[\Gamma; \Delta \vdash T]$
- Base case is inductive on r.h.s types:
  - $P \in \mathcal{L}[z{:}A \multimap B] \triangleq$ if $P \overset{z(y)}{\Rightarrow} P'$ then $\forall Q \in \mathcal{L}[y{:}A].(\nu y)(P' \mid Q) \in \mathcal{L}[z{:}B]$
  - ...
- Show that well-typed processes are in the predicate.

Process Equivalence

- Equivalence is a fundamental tool for reasoning about processes.
- “Canonical” observational equivalence: Barbed Congruence.
- “Hard” to reason about – Quantification over all contexts.

Reasoning Techniques

Process Equivalence

- Equivalence is a fundamental tool for reasoning about processes.
- “Canonical” observational equivalence: Barbed Congruence.
- “Hard” to reason about – Quantification over all contexts.

Logical Equivalence

- Natural extension of unary predicate.
- Contextual typed bisimulation – Logical Equivalence.
- “Easy” to reason about.
- Logical Equivalence vs Barbed Congruence:
    - Propositional: Sound and Complete
    - Polymorphic: Sound and Complete
    - Higher-order: ???

(Functional) Type Theory

- Types depend on terms.
- Uniform integration of programming and reasoning.
- Truly, *Proofs as Programs*.

(Functional) Type Theory

- Types depend on terms.
- Uniform integration of programming and reasoning.
- Truly, *Proofs as Programs*.

Towards Dependent Session Types

- Value dependencies are insufficient.
- Separation of data and behavior – only specify properties of data.
- Monadic language “closes the gap”.
- Restrict to logically sound fragment.
- Coinduction is crucial when reasoning about processes.

Reasoning Techniques

Dependent Session Types

Challenges

- Understanding definitional equality.
- Rich enough for practical proofs? Decidable?
- Adequacy

Potential Outcome – Coinductive Proofs as Processes

```
equiv : {c:natStream} -> {c:natStream} -> stype.
stype bisim P Q = ...

S1 : nat -> {c:natStream}
S2 : nat -> {c:natStream}

bisim_proof : Pi n:nat.{c:bisim (S1 n) (S2 n)}
c <- bisim_proof n = { ... }
```

Conclusion

Summary

- Logical foundation of session types using linear logic:
  - Concurrent language features explained logically.
  - Logical features explained using concurrency.
- A concurrent programming language:
  - Monadic integration of session-based concurrency.
  - General recursion and (re)connections with logic.
- Reasoning techniques:
  - Non-trivial properties (e.g. Termination, Parametricity).
  - Behavioral equivalence.

Conclusion

Summary

- Logical foundation of session types using linear logic:
  - Concurrent language features explained logically.
  - Logical features explained using concurrency.
- A concurrent programming language:
  - Monadic integration of session-based concurrency.
  - General recursion and (re)connections with logic.
- Reasoning techniques:
  - Non-trivial properties (e.g. Termination, Parametricity).
  - Behavioral equivalence.

Research Plan (Approx. 6 months)

- Study behavioral theory in HO setting.
- Definitional equality and dependent session types (speculative).
- Develop examples showcasing expressiveness.

Bibliography

- B. Toninho, L. Caires, F. Pfenning. *Dependent Session Types via Intuitionistic Linear Type Theory*. PPDP'11
- F. Pfenning, L. Caires, B. Toninho. *Proof-Carrying Code in a Session-Typed Process Calculus*. CPP'11
- J. Pérez, L. Caires, F. Pfenning, B. Toninho. *Linear Logical Relations for Session-Based Concurrency*. ESOP'12
- B. Toninho, L. Caires, F. Pfenning. *Functions as Session-Typed Processes*. FoSSaCS'12
- L. Caires, J. Pérez, F. Pfenning, B. Toninho. *Behavioral Polymorphism and Parametricity in Session-Based Communication*. ESOP'13
- B. Toninho, L. Caires, F. Pfenning. *Higher-Order Processes, Functions, and Sessions: A Monadic Integration*. ESOP'13
- B. Toninho, L. Caires, F. Pfenning. *Corecursion and Non-Divergence in Session Types*. TGC'14 (to appear)

A Logical Foundation for Session-based Concurrent Computation

Bernardo Toninho
[Co-advised by: Luís Caires and Frank Pfenning]

Computer Science Department
Carnegie Mellon University
Universidade Nova de Lisboa

Thesis Proposal