Session Types and Linear Logic

Bernardo Toninho and Nobuko Yoshida

University of Bath,
November 28, 2017
Us ∈ Mobility Research Group

http://mrg.doc.ic.ac.uk/
• **TCS’16**: Monitoring Networks through Multiparty Session Types. Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, Nobuko Yoshida
• **LMCS’16**: Multiparty Session Actors. Rumiya Neykova, Nobuko Yoshida
• **FMSD’15**: Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon, Kohei Honda, Raymond Hu, Rumiya Neykova, Nobuko Yoshida
• **TGC’13**: The Scribble Protocol Language. Nobuko Yoshida, Raymond Hu, Rumiya Neykova, Nicholas Ng
Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.

Describe 🖋
Scribble is a language for describing multiparty protocols from a global, or endpoint neutral, perspective.

Verify 🎈
Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and do not suffer from deadlocks or livelocks.

Project ⚙️
Endpoint projection is the term used for identifying the responsibility of a particular role (or endpoint) within a protocol.

Implement 📃
Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c) statically verify the code against the projection.

Monitor 🛡️
Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of a particular endpoint, to ensure it correctly implements the expected behaviour.
module examples;

global protocol HelloWorld(role Me, role World) {
  hello() from Me to World;
  choice at World {
    goodMorning1() from World to Me;
  } or {
    goodMorning1() from World to Me;
  }
}
End-to-End Switching Programme by DCC

1. All design work takes place in ABACUS, DCC’s enterprise architecture tool. This can export standard XMI files (an open standard for UML5)

2. XMI is converted into OpenTracing format for consumption by managed service

3. OpenTracing files are combined to build a model in Scribble

4. Model holds types rather than instances to understand behaviour

5. Scribble compiler identifies inconsistency, change & design flaws

6. Issues highlighted graphically in Eclipse

7. Generate exception report and send back to DCC

www.estafet.com Estafet Managed Service
End-to-End Switching Programme by DCC

Caveats:
1. Using earlier implementation of Scribble (CDL), because we already have those tools
2. Using earlier plugin to Eclipse - we’d want to improve this
3. We’re not going via OpenTracing - this is part of the bid costs

Scope of the demo:
7. Generate exception report and send back to DCC

3. OpenTracing files are combined to build a model in Scribble
4. Model holds types rather than instances to understand behaviour
5. Scribble compiler identifies inconsistency, change & design flaws
6. Issues highlighted graphically in Eclipse

www.estafet.com

Estafet Managed Service
Interactions with Industries

Strange Loop
SEPTEMBER 15-17 2016 / PEABODY OPERA HOUSE / ST. LOUIS, MO

Adam Bowen @adamnbowen • Sep 15
I didn’t even know that session types existed an hour ago, but thanks to Nobuko Yoshida’s great talk at #pwiconf, I want to learn more.

Nobuko Yoshida
Imperial College, London

DoC researcher to speak at Golang UK conference
by Vicky Kapogianni
20 July 2016

DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research

@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16

The Golang UK Conference
Interactions with Industries

F#unctional Londoners Meetup Group

6 days ago · 6:30 PM
Session Types with Fahd Abdeljallal

43 Members

Synopsis: Session types are a formalism to codify the structure of a communication, using types to specify the communication protocol used. This formalism provides the... LEARN MORE

Distributed Systems vs. Compositionality

Dr. Roland Kuhn
@rolandkuhn — CTO of Actyx

Current State

- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)
Go concurrency verification research at DoC grabs headline

A paper by DoC researchers at POPL on Go concurrency verification was featured in a tech blog and generates a buzz outside of the research community.

A paper by researchers at the department was recently featured in the morning paper, a blog by venture capitalist Adrian Colye, which summarises an important, influential, topical or otherwise interesting paper in the field of computer science every weekday in an easily digestible way by non-researchers. On the 2 Feb 2017 issue of the morning paper, it was highlighted as "the true spirit of POPL (Principles of Programming Languages)".
Selected Publications 2016/2017


• [CC’16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.


• [POPL’16] Dominic Orchard, NY: Effects as sessions, sessions as effects.
• [CC’16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
Session Types and Linear Logic

Bernardo Toninho and Nobuko Yoshida

University of Bath, November 28, 2017
Introduction
Concurrent Processes

- Coordination of multiple simultaneously executing agents.
- Programming Models: Shared-Memory vs Message-Passing
- Hard to reason about:
  - Deadlocks
  - Data races
  - Concurrent interleavings make behaviour hard to predict
  - ... and also hard to replicate (e.g. testing).
Introduction
A Concurrency-Theoretic Approach: Session Types

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.”
Introduction
A Concurrency-Theoretic Approach: Session Types

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.”

Session-based Structuring
- Specify communication behaviour as sessions.
- Check that programs adhere to specification (session fidelity).
Session Types [Honda93]

- Types are descriptions of communication behaviour, assigned to channels.
- A way of guaranteeing communication discipline, statically.
- A linear typing discipline for $\pi$-calculus processes.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select
**Session Types [Honda93]**

- Types are descriptions of communication behaviour, assigned to channels.
- A way of guaranteeing communication discipline, **statically**.
- A linear typing discipline for $\pi$-calculus processes.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select

**Syntax**

\[
P, Q ::= x(y).P \mid x(y).P \mid x.\ell; P \mid x.\text{case}\{\ell_i : P_i\}_{i\in I} \mid (\nu x)P \mid 0 \mid (P \mid Q)
\]

\[
S, T ::= !S.T \mid ?S.T \mid \oplus \{\ell_i : T_i\}_{i\in I} \mid \& \{\ell_i : T_i\}_{i\in I}
\]
Introduction
A Concurrency-Theoretic Approach: Session Types

Session Types [Honda93]
- Types are descriptions of communication behaviour, assigned to channels.
- A way of guaranteeing communication discipline, statically.
- A linear typing discipline for \( \pi \)-calculus processes.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select

Syntax

\[
P, Q ::= x(y).P \mid x(y).P \mid x.\ell; P \mid x.\text{case}\{\ell_i : P_i\}_{i \in I} \mid (\nu x)P \mid 0 \mid (P \mid Q)
\]

\[
S, T ::= !S.T \mid ?S.T \mid \oplus \{\ell_i : T_i\}_{i \in I} \mid \& \{\ell_i : T_i\}_{i \in I}
\]
Introduction
A Concurrency-Theoretic Approach: Session Types

Session Types [Honda93]
- Types are descriptions of communication behaviour, assigned to channels.
- A way of guaranteeing communication discipline, statically.
- A linear typing discipline for $\pi$-calculus processes.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select

Syntax

\[
\begin{align*}
P, Q & ::= x\langle y \rangle . P \mid x(y). P \mid x. \ell; P \mid x. \text{case}\{\ell_i : P_i\}_{i \in I} \mid (\nu x) P \mid 0 \mid (P \mid Q) \\
S, T & ::= !S.T \mid ?S.T \mid \oplus \{\ell_i : T_i\}_{i \in I} \mid \& \{\ell_i : T_i\}_{i \in I}
\end{align*}
\]
Introduction
A Concurrency-Theoretic Approach: Session Types

Session Types [Honda93]
- Types are descriptions of communication behaviour, assigned to channels.
- A way of guaranteeing communication discipline, statically.
- A linear typing discipline for $\pi$-calculus processes.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select

Syntax

$P, Q ::= x\langle y \rangle . P \mid x(y) . P \mid x . \ell ; P \mid x . \text{case} \{ \ell_i : P_i \}_{i \in I} \mid (\nu x)P \mid 0 \mid (P \mid Q)$

$S, T ::= !S.T \mid ?S.T \mid \oplus \{ \ell_i : T_i \}_{i \in I} \mid \& \{ \ell_i : T_i \}_{i \in I}$
Introduction
A Concurrency-Theoretic Approach: Session Types

Session Types [Honda93]
- Types are descriptions of communication behaviour, assigned to channels.
- A way of guaranteeing communication discipline, statically.
- A linear typing discipline for $\pi$-calculus processes.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select

Syntax
\[
P, Q ::= \ x\langle y\rangle . P \mid x(y). P \mid x. \ell; P \mid x. \text{case}\{\ell_i : P_i\}_{i \in I} \mid (\nu x) P \mid 0 \mid (P \mid Q) \\
S, T ::= !S.T \mid ?S.T \mid \bigoplus\{\ell_i : T_i\}_{i \in I} \mid \&\{\ell_i : T_i\}_{i \in I}
\]

A pair of processes interacting on dual sessions is \textit{deadlock-free}!
Introduction
Session Types and Linear Logic

Linear Logic [Girard87]
- A marriage of classical dualities and constructivism.
- A logic of resources and interaction.
- Resource independence captures non-determinism/concurrency.
- Connected to concurrency early on [Abramsky93,BellinScott94]
Introduction
Session Types and Linear Logic

Linear Logic [Girard87]
- A marriage of classical dualities and constructivism.
- A logic of resources and interaction.
- Resource independence captures non-determinism/concurrency.
- Connected to concurrency early on [Abramsky93,BellinScott94]

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

Why does it matter?

» Good metalogical properties map to good program properties.
» Progress, session fidelity, type preservation “for free”.
» Reasoning built-in.
» Much more compositional than traditional approaches (i.e. extensibility).
Introduction

Roadmap

1. Basic interpretation
2. Enriching the type structure (dependent types and polymorphism)
ILL Interpretation

Key Ideas

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

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

**Propositions**

$$A, B ::= A \otimes B | A \rightarrow B | 1 | A \& B | A \oplus B | !A$$
### Key Ideas

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

### Propositions

\[ A, B \ ::= \ A \otimes B \mid A \to B \mid 1 \mid A \& B \mid A \oplus B \mid !A \]

### Sequents

- Duality of offering (right rules) and using (left rules) a session.
- Proof composition (cut) as process composition.
- Identity as forwarding/renaming.
ILL Interpretation
Judgmental Principles

Typing Judgment

\[ \Gamma : A_1, \ldots, A_m; \quad \Delta : A_1, \ldots, A_n \Rightarrow A \]
Typing Judgment

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

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

**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**

\[ \Gamma; \Delta \Rightarrow A \quad \Gamma; \Delta', A \Rightarrow C \quad \Gamma; \Delta, \Delta' \Rightarrow C \text{ cut} \]
ILL Interpretation
Judgmental Principles

**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**

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

\[ \Gamma ; \Delta , \Delta' \Rightarrow C \]  \hspace{1cm} \text{cut}
ILL Interpretation
Judgmental Principles

Typing Judgment

\[ \Gamma; \Delta \vdash \Gamma; \Delta = \Rightarrow P :: x:A \]

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

Cut as Composition

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

\[ \Gamma; \Delta, \Delta' \Rightarrow C \quad \text{cut} \]
ILL Interpretation
Judgmental Principles

Typing Judgment

\[
\begin{aligned}
\Gamma &\vdash u_1 : A_1, \ldots, u_m : A_m; x_1 : A_1, \ldots, x_n : A_n \Rightarrow P :: x : A \\
\Delta &\vdash
\end{aligned}
\]

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

Cut as Composition

\[
\begin{aligned}
\Gamma; \Delta &\Rightarrow P :: x : A \\
\Gamma; \Delta', x : A &\Rightarrow Q :: z : C
\end{aligned}
\]

\[
\begin{aligned}
\Gamma; \Delta, \Delta' &\Rightarrow \nu x (P \mid Q) :: z : C
\end{aligned}
\]

\text{cut}
ILL Interpretation
Judgmental Principles

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

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

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

cut

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

Identity as Renaming

\[ \Gamma; A \Rightarrow A \]
ILL Interpretation
Judgmental Principles

Typing Judgment

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

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

Identity as Renaming

\[
\Gamma; x: A \quad \Rightarrow \quad [x \leftrightarrow z] :: z: A
\]
ILL Interpretation

Propositions

**Multiplicative Conjunction**

\[
\frac{\Gamma; \Delta \Rightarrow A \quad \Gamma; \Delta' \Rightarrow B}{\Gamma; \Delta, \Delta' \Rightarrow A \otimes B} \quad \otimes R
\]
ILL Interpretation

Propositions

Multiplicative Conjunction

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

\[
\Gamma; \Delta, \Delta' \Rightarrow \quad \Omega \quad R
\]

\[
A \otimes B
\]
ILL Interpretation

Propositions

Multiplicative Conjunction

\[
\begin{align*}
\Gamma; \Delta & \quad \Rightarrow \quad P_1 :: y : A \\
\Gamma; \Delta' & \quad \Rightarrow \quad P_2 :: x : B \\
\Gamma; \Delta, \Delta' & \quad \Rightarrow \quad (\nu y) x \langle y \rangle . (P_1 | P_2) :: x : A \otimes B
\end{align*}
\]
ILL Interpretation

Propositions

**Multiplicative Conjunction**

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

\[
\frac{\Gamma; \Delta, A, B \rightarrow C}{\Gamma; \Delta, A \otimes B \rightarrow C} \quad \otimes L
\]
ILL Interpretation

Propositions

### Multiplicative Conjunction

\[
\begin{align*}
\Gamma; \Delta & \rightarrow P_1 :: y : A & \Gamma; \Delta' & \rightarrow P_2 :: x : B \\
\Gamma; \Delta, \Delta' & \rightarrow (\nu y) (x \langle y \rangle) \cdot (P_1 | P_2) :: x : A \otimes B \\
\Gamma; \Delta, y : A, x : B & \rightarrow Q :: z : C \\
\Gamma; \Delta, A \otimes B & \rightarrow C \otimes L
\end{align*}
\]
ILL Interpretation
Propositions

Multiplicative Conjunction

\[
\begin{align*}
\Gamma; \Delta & \Rightarrow P_1 :: y: A & \Gamma; \Delta' & \Rightarrow P_2 :: x: B \\
\Gamma; \Delta, \Delta' & \Rightarrow (\nu y)x\langle y \rangle.(P_1 \mid P_2) :: x:A \otimes B
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta, y : A, x : B & \Rightarrow Q :: z: C \\
\Gamma; \Delta, x: A \otimes B & \Rightarrow x(y).Q :: z: C
\end{align*}
\]
### ILL Interpretation

#### Propositions

**Multiplicative Conjunction**

\[
\Gamma; \Delta \rightarrow P_1 :: y:A \quad \Gamma; \Delta' \rightarrow P_2 :: x:B \quad \boxed{\otimes R}
\]

\[
\Gamma; \Delta, \Delta' \rightarrow (\nu y)x\langle y\rangle.(P_1 | P_2) :: x:A \otimes B
\]

\[
\Gamma; \Delta, y : A, x : B \rightarrow Q :: z:C \\
\boxed{\otimes L}
\]

\[
\Gamma; \Delta, x:A \otimes B \rightarrow x(y).Q :: z:C
\]

**Proof Reduction**

\[
\Gamma; \Delta, \Delta' \rightarrow (\nu x)((\nu y)x\langle y\rangle.(P_1 | P_2) | x(y).Q) :: z:C
\]
ILL Interpretation

Propositions

Multiplicative Conjunction

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

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

Proof Reduction

\[\Gamma; \Delta, \Delta' \rightarrow (\nu x)((\nu y)x\langle y\rangle.(P_1 | P_2) | x(y).Q) :: z:C\]
\[\rightarrow \Gamma; \Delta, \Delta' \rightarrow (\nu x)(\nu y)(P_1 | P_2 | Q) :: z:C\]
ILL Interpretation
Propositions

Linear Implication

\[ \Gamma; \Delta, \quad A \implies B \quad \quad \quad \quad \quad \Gamma; \Delta \implies A \leadsto B \quad \quad \quad \quad \quad \neg R \]
ILL Interpretation

Propositions

Linear Implication

\[ \Gamma; \Delta, y : A \Rightarrow P :: x : B \]

\[ \Gamma; \Delta \Rightarrow A \leftarrow B \quad \xrightarrow{\bullet} R \]
ILL Interpretation

Propositions

Linear Implication

\[ \Gamma; \Delta, y : A \Longrightarrow P :: x:B \]
\[ \Gamma; \Delta \Longrightarrow x(y).P :: x:A \leadsto B \]
Linear Implication

\[
\begin{align*}
\Gamma; \Delta, y : A & \Rightarrow P :: x : B \\
\Gamma; \Delta & \Rightarrow x(y).P :: x : A \Rightarrow B & \xrightarrow{\Rightarrow} R \\
\Gamma; \Delta & \Rightarrow A & \Gamma; \Delta' & \Rightarrow B & \Rightarrow C & \xrightarrow{\Rightarrow} L
\end{align*}
\]
**ILL Interpretation**

**Propositions**

<table>
<thead>
<tr>
<th>Linear Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma; \Delta, y : A \Rightarrow P :: x : B ]</td>
</tr>
<tr>
<td>[ \Rightarrow \Gamma; \Delta \Rightarrow x(y).P :: x : A \leadsto B \rightarrow R ]</td>
</tr>
<tr>
<td>[ \Gamma; \Delta \Rightarrow Q_1 :: y : A \Gamma; \Delta', x : B \Rightarrow Q_2 :: z : C ]</td>
</tr>
<tr>
<td>[ \Rightarrow \Gamma; \Delta, \Delta', A \leadsto B \leadsto C \rightarrow L ]</td>
</tr>
</tbody>
</table>
Linear Implication

\[
\frac{\Gamma; \Delta, y : A \Rightarrow P :: x:B}{\Gamma; \Delta \Rightarrow x(y).P :: x:A \multimap B} \quad \text{←} \circ R
\]

\[
\frac{\Gamma; \Delta \Rightarrow Q_1 :: y:A \quad \Gamma; \Delta', x:B \Rightarrow Q_2 :: z:C}{\Gamma; \Delta, \Delta', x:A \multimap B \Rightarrow (\nu y)x\langle y\rangle.(Q_1 | Q_2) :: z:C} \quad \text{←} \circ L
\]
ILL Interpretation

Propositions

Linear Implication

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

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

Linear Implication as input. Reduction is the same as for \(\otimes\).
ILL Interpretation

Propositions

Multiplicative Unit

\[ \Gamma; \cdot \quad 1 \quad 1R \]

Proof Reduction

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

Bernardo Toninho
ILL Interpretation

Propositions

Multiplicative Unit

\[ \Gamma; \cdot \rightarrow 0 :: x:1 \]

\[ 1R \]
ILL Interpretation
Propositions

Multiplicative Unit

\[
\frac{\Gamma; \cdot \rightarrow 0 :: x:1}{1R} \quad \frac{\Gamma; \Delta \rightarrow C}{\Gamma; \Delta, \ 1 \rightarrow C \quad 1L}
\]
ILL Interpretation

Propositions

Multiplicative Unit

\[ \Gamma; \cdot \rightarrow 0 :: x:1 \quad 1R \]
\[ \Gamma; \Delta \rightarrow Q :: z:C \]
\[ \Gamma; \Delta, 1 \rightarrow C \quad 1L \]
### ILL Interpretation

#### Propositions

<table>
<thead>
<tr>
<th>Multiplicative Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma; \cdot \Rightarrow 0 :: x:1 )</td>
</tr>
<tr>
<td>1R</td>
</tr>
<tr>
<td>( \Gamma; \Delta \Rightarrow Q :: z:C )</td>
</tr>
<tr>
<td>1L</td>
</tr>
<tr>
<td>( \Gamma; \Delta, x:1 \Rightarrow Q :: z:C )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma; \Delta \Rightarrow (\nu x)(0</td>
</tr>
</tbody>
</table>
ILL Interpretation

Propositions

**Multiplicative Unit**

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

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

\[
\Gamma; \Delta, x:1 \Rightarrow Q :: z:C
\]

**Proof Reduction**

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

\[
\equiv \quad \Gamma; \Delta \Rightarrow Q :: z:C
\]
ILL Interpretation
Propositions

Multiplicative Unit

\[
\frac{\Gamma; \cdot \rightarrow 0 :: x:1}{1R} \quad \frac{\Gamma; \Delta \rightarrow Q :: z:C}{1L}
\]

Proof Reduction

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

Multiplicative Unit as Termination
ILL Interpretation

Propositions

Additive Conjunction

\[
\frac{\Gamma; \Delta \Rightarrow A \quad \Gamma; \Delta \Rightarrow B}{\Gamma; \Delta \Rightarrow A \& B} \quad \& R
\]
ILL Interpretation

Propositions

Additive Conjunction

\[
\begin{align*}
\Gamma; \Delta & \Rightarrow P_1 :: x:A & \Gamma; \Delta & \Rightarrow P_2 :: x:B \\
\Gamma; \Delta & \Rightarrow & A \& B & \& \mathcal{R}
\end{align*}
\]
Additive Conjunction

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

\&R
# ILL Interpretation

## Propositions

### Additive Conjunction

\[
\begin{align*}
&\Gamma; \Delta \Longrightarrow P_1 :: x:A & \Gamma; \Delta \Longrightarrow P_2 :: x:B \\
&\Gamma; \Delta \Longrightarrow x.\text{case}(P_1, P_2) :: x:A \& B
\end{align*}
\]

& R

\[
\begin{align*}
&\Gamma; \Delta, \quad A \Longrightarrow C \\
&\Gamma; \Delta, \quad A \& B \Longrightarrow C
\end{align*}
\]

& L_1

---

Bernardo Toninho

mrg.doc.ic.ac.uk
ILL Interpretation

Additive Conjunction

\[
\begin{align*}
\Gamma; \Delta \Rightarrow & \quad P_1 :: x:A & \quad \Gamma; \Delta \Rightarrow & \quad P_2 :: x:B \\
\Gamma; \Delta \Rightarrow & \quad x.\text{case}(P_1, P_2) :: x:A \land B & \quad \land R
\end{align*}
\]

\[
\Gamma; \Delta, x:A \Rightarrow & \quad Q :: z:C \\
\Gamma; \Delta, A \land B \Rightarrow & \quad C & \quad \land L_1
\]

Bernardo Toninho mrg.doc.ic.ac.uk 13/32
ILL Interpretation
Propositions

Additive Conjunction

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

\& R

\[
\begin{align*}
\Gamma; \Delta, x:A \rightarrow Q :: z:C
\end{align*}
\]

\& L_1

\[
\begin{align*}
\Gamma; \Delta, x:A \& B \rightarrow x.\text{inl}; Q :: z:C
\end{align*}
\]
ILL Interpretation

Propositions

Additive Conjunction

\[
\begin{align*}
\Gamma; \Delta & \implies P_1 :: x:A & \Gamma; \Delta & \implies P_2 :: x:B \\
\Gamma; \Delta & \implies x.\text{case}(P_1, P_2) :: x:A \land B \\
\Gamma; \Delta, x:A & \implies Q :: z:C \\
\Gamma; \Delta, x:A \land B & \implies x.\text{inl}; Q :: z:C
\end{align*}
\]

\&R

&L_1

Proof Reduction

\[
\begin{align*}
\Gamma; \Delta, \Delta' & \implies (\nu x)(x.\text{case}(P_1, P_2) \mid x.\text{inl}; Q) :: z:C \\
& \implies \Gamma; \Delta, \Delta' \implies (\nu x)(P_1 \mid Q) :: z:C
\end{align*}
\]
ILL Interpretation

Propositions

Additive Disjunction

\[
\frac{\Gamma; \Delta \Rightarrow A \oplus B}{\Gamma; \Delta \Rightarrow A \oplus B} \quad \oplus R_1
\]
ILL Interpretation

Propositions

Additive Disjunction

\[
\Gamma; \Delta \Rightarrow P :: x : A \\
\Gamma; \Delta \Rightarrow A \oplus B \quad \oplus R_1
\]
ILL Interpretation

Propositions

Additive Disjunction

\[
\Gamma; \Delta \Rightarrow P :: x:A \\
\Gamma; \Delta \Rightarrow x.\text{inl}; P :: x:A \oplus B \quad \oplus R_1
\]
**ILL Interpretation**

**Propositions**

### Additive Disjunction

\[
\Gamma; \Delta \Rightarrow P : x : A \\
\Gamma; \Delta \Rightarrow x. \text{inl}; P : x : A \oplus B \\
\Gamma; \Delta \Rightarrow x. \text{case} (Q_1, Q_2) : z : C \oplus L
\]

Same proof reductions as \( R_1 \).
ILL Interpretation

Propositions

Additive Disjunction

\[ \Gamma; \Delta \Rightarrow P :: x: A \]
\[ \Gamma; \Delta \Rightarrow x.\text{inl}; P :: x: A \oplus B \quad \oplus R_1 \]

\[ \Gamma; \Delta, x:A \Rightarrow Q_1 :: z:C \quad \Gamma; \Delta, x:B \Rightarrow Q_2 :: z:C \]
\[ \Gamma; \Delta, A \oplus B \Rightarrow C \quad \oplus L \]
ILL Interpretation
Propositions

Additive Disjunction

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

Same proof reductions as \&.
### Persistent Cut

\[
\begin{array}{c}
\Gamma; \cdot \Rightarrow A \\
\Gamma, A; \Delta \Rightarrow C \\
\end{array}
\]

\[
\Gamma; \Delta \Rightarrow C \quad \text{cut}^\dagger
\]
ILL Interpretation
Judgmental Principles for Exponential

**Persistent Cut**

\[
\frac{\Gamma; \cdot \Rightarrow P :: x:A \quad \Gamma, u:A; \Delta \Rightarrow Q :: z:C}{\Gamma; \Delta \Rightarrow \quad \text{cut}^l}
\]

\[
\frac{\nu u (x) \langle x \rangle.P | \nu x (u x) (P | Q)) \Rightarrow z:C}{\Rightarrow \Gamma; \Delta \Rightarrow}
\]
ILL Interpretation
Judgmental Principles for Exponential

Persistent Cut

\[
\frac{\Gamma; \cdot \Rightarrow P :: x: A \quad \Gamma, u: A; \Delta \Rightarrow Q :: z: C}{\Gamma; \Delta \Rightarrow (\nu u) (\!u(x). P \mid Q) :: z: C}
\]

\text{cut}^l
ILL Interpretation
Judgmental Principles for Exponential

Persistent Cut

$$\Gamma; \cdot \Rightarrow P :: x:A \quad \Gamma, u:A; \Delta \Rightarrow Q :: z:C$$

$$\Gamma; \Delta \Rightarrow (\nu u)(!u(x).P \mid Q) :: z:C$$

Parallel composition of $P$, offering $x:A$ and $Q$, using $u:A$ persistently.
ILL Interpretation
Judgmental Principles for Exponential

Persistent Cut

\[ \Gamma; \cdot \Rightarrow P :: x : A \quad \Gamma, u : A; \Delta \Rightarrow Q :: z : C \]
\[ \Gamma; \Delta \Rightarrow (\nu u)(!u(x). P \mid Q) :: z : C \]
\[ \text{cut}^l \]

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

Copy

\[ \Gamma, A; \Delta, A \Rightarrow C \]
\[ \Gamma, A; \Delta \Rightarrow C \]
\[ \text{copy} \]
**ILL Interpretation**

Judgmental Principles for Exponential

### Persistent Cut

\[ \Gamma; \cdot \quad \rightarrow \quad P :: x : A \quad \Gamma, u : A; \Delta \quad \rightarrow \quad Q :: z : C \]

\[ \Gamma; \Delta \quad \rightarrow \quad (\nu u)(!u(x).P \mid Q) :: z : C \]

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

### Copy

\[ \Gamma, u : A; \Delta, x : A \quad \rightarrow \quad P :: z : C \]

\[ \Gamma, A; \Delta \quad \rightarrow \quad C \]

Copy
ILL Interpretation
Judgmental Principles for Exponential

**Persistent Cut**

\[
\frac{\Gamma; \cdot \Rightarrow P :: x:A \quad \Gamma, u:A; \Delta \Rightarrow Q :: z:C}{\Gamma; \Delta \Rightarrow (\nu u)(!u(x).P | Q) :: z:C} \quad \text{cut}^l
\]

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

**Copy**

\[
\frac{\Gamma, u:A; \Delta, x:A \Rightarrow P :: z:C}{\Gamma, u:A; \Delta \Rightarrow (\nu x)u(x).Q :: z:C} \quad \text{copy}
\]
ILL Interpretation
Judgmental Principles for Exponential

**Persistent Cut**

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

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

**Copy**

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

**Proof Reduction**

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

Propositions

Exponential

\[
\frac{\Gamma; \cdot \Rightarrow A \quad \Gamma; \cdot \Rightarrow !A}{\Gamma; \cdot \Rightarrow !R}
\]
ILL Interpretation

Propositions

Exponential

\[ \Gamma; \cdot \Rightarrow P :: y : A \]

\[ \Gamma; \cdot \Rightarrow !A \quad !R \]

\[ \Gamma; \cdot \Rightarrow !A \]
ILL Interpretation
Propositions

Exponential

\[
\Gamma; \cdot \Rightarrow P :: y : A \\
\Gamma; \cdot \Rightarrow !x(y).P :: x : !A \\
\Rightarrow !R
\]
ILL Interpretation

Propositions

Exponential

\[ \frac{}{\Gamma; \cdot \Rightarrow P :: y:A} \]

\[ \frac{}{\Gamma; \cdot \Rightarrow \textbf{!}x(y).P :: x:!A} \]

\[ \frac{}{\textbf{!}R} \]

\[ \frac{}{\Gamma, \ A; \Delta \Rightarrow C} \]

\[ \frac{}{\Gamma; \Delta, \ !A \Rightarrow} \]

\[ \frac{}{C \ \textbf{!}L} \]
ILL Interpretation

Propositions

Exponential

\[
\Gamma; \cdot \Rightarrow P :: y:A \\
\Gamma; \cdot \Rightarrow !x(y).P :: x:!A \\
\Gamma; \cdot \Rightarrow !R \\
\Gamma, u:A; \Delta \Rightarrow P :: z:C \\
\Gamma; \Delta, !A \Rightarrow C \\
\Gamma; \Delta, !L
\]
ILL Interpretation

Propositions

Exponential

\[
\Gamma; \cdot \rightarrow P :: y:A \\
\Gamma; \cdot \rightarrow \!x(y).P :: x:A \\
\frac{}{\Gamma; \cdot \rightarrow \!x(y).P :: x:A} \quad \frac{}{\Gamma, u:A; \Delta \rightarrow P :: z:C} \\
\frac{}{\Gamma, u:A; \Delta \rightarrow P :: z:C} \quad \frac{}{\Gamma, \Delta, x:A \rightarrow P\{x/u\} :: z:C} \\
\frac{}{\Gamma, \Delta, x:A \rightarrow P\{x/u\} :: z:C} \\
\]

Proof reduction transforms a cut into a cut\(^1\) (struct. equivalence).
ILL Interpretation

Metatheory

Operational Correspondence and Subject Reduction

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

Metatheory

**Operational Correspondence and Subject Reduction**

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

**Global Progress**

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

If $\emptyset \rightarrow P : x:1$ and live$(P)$ then $\exists Q$ such that $P \rightarrow Q$.

Much stronger property than in classical session types, “for free”!

Bernardo Toninho
ILL Interpretation
Summary

- Linear Propositions as Session Types.
- Intuitionistic proofs as session-typed processes.
- Process reduction maps to proof conversion.
Linear Propositions as Session Types.
Intuitionistic proofs as session-typed processes.
Process reduction maps to proof conversion.
... but not all proof conversions are process reductions!
Richer Type Theories
Motivation

Session Types
- Only express simple communication patterns.
- No interesting properties of exchanged data.
- No sophisticated properties of processes.
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 Session Types [Toninho et al.11]
- Polymorphic Session Types [Pérez et al.11, Wadler11]
- Internalisation in a linear monad [Toninho et al.12]
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 $\exists x: \tau. A$
- Parametric in the language of types $\tau$. 
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 $\exists x:\tau.A$
- 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 $A(M)$. 
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 $\exists x: \tau. A$
- 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.
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 $\exists x: \tau. A$
- 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.

\[
\text{indexer}_{\text{simple}} \triangleq !(\text{file} \to \text{file} \otimes 1)
\]
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 $\exists x : \tau. A$
- 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.

```
indexer_{\text{simple}} \triangleq ! (\text{file} \rightarrow \text{file} \otimes 1)
indexer_{\text{dspec}} \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?

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

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

\[
\text{indexer}_{simple} \triangleq ! (\text{file} \to \text{file} \otimes 1) \\
\text{indexer}_{dspec} \triangleq ! (\forall f:\text{file.pdf}(f) \to \exists g:\text{file.pdf}(g) \otimes \text{agree}(f, g) \otimes 1) \\
\text{indexer}_{irrev} \triangleq ! (\forall f:\text{file.[pdf}(f)] \to \exists g:\text{file.[pdf}(g)] \otimes [\text{agree}(f, g)] \otimes 1)
\]
Richer Type Theories
Where are we?

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

- Second-order quantification ($\forall X.A$ and $\exists X.A$).
- Communication of session types / abstract protocols.
- Relational parametricity results in the style of System F.
Richer Type Theories
Where are we?

Polymorphism and Parametricity [Pérez et al.11, Wadler11]
- Second-order quantification ($\forall X.A$ and $\exists X.A$).
- Communication of session types / abstract protocols.
- Relational parametricity results in the style of System F.

Monadic Integration [Toninho et al.12]
- $M : \{\Gamma; \Delta \vdash z : A\}$, functional type of an open process $P$ (construct $\{P\}$).
- Process construct bind$(M, z.Q)$:
  - Evaluate $M$ to a (suspended) process $\{P\}$.
  - Run underlying process $P$ in parallel with $Q$. 
Richer Type Theories
Where are we?

Polymorphism and Parametricity [Pérez et al.11, Wadler11]
- Second-order quantification ($\forall X.A$ and $\exists X.A$).
- Communication of session types / abstract protocols.
- Relational parametricity results in the style of System F.

Monadic Integration [Toninho et al.12]
- $M : \{\Gamma; \Delta \vdash z:A\}$, functional type of an open process $P$ (construct $\{P\}$).
- Process construct bind($M, z.Q$):
  - Evaluate $M$ to a (suspended) process $\{P\}$.
  - Run underlying process $P$ in parallel with $Q$.
- Higher-Order Sessions, e.g.: $\forall x: \{z:A\}. \forall y: \{z:B\}. A \otimes B$
Richer Type Theories
Where are we?

Monadic Dependencies (Ongoing work)

By adding the (contextual) monad to a dependent type theory we get:

- Relations on processes as type families (e.g. \( \Pi x: \{ z:A \}. \Pi y: \{ z:B \}. \text{type} \))
Richer Type Theories

Where are we?

Monadic Dependencies (Ongoing work)

By adding the (contextual) monad to a dependent type theory we get:

- Relations on processes as type families (e.g. $\Pi x: \{z:A\}. \Pi y: \{z:B\}. \text{type}$)
- Value dependent communication (e.g. $\forall x: \text{Nat}. \text{if } x = 5 \text{ then } T_1 \text{ else } T_2$)
Monadic Dependencies (Ongoing work)

By adding the (contextual) monad to a dependent type theory we get:

- Relations on processes as type families (e.g. $\Pi x: \{z:A\}. \Pi y: \{z:B\}. \text{type}$)
- Value dependent communication (e.g. $\forall x: \text{Nat}. \text{if } x = 5 \text{ then } T_1 \text{ else } T_2$)
- Indexed session types: $\text{countDown} :: \Pi x: \text{Nat}. \text{stype}$
Richer Type Theories
Where are we?

Monadic Dependencies (Ongoing work)

By adding the (contextual) monad to a dependent type theory we get:

- Relations on processes as type families (e.g. $\Pi x: \{z:A\}. \Pi y: \{z:B\}. \text{type}$)
- Value dependent communication (e.g. $\forall x: \text{Nat}. \text{if } x = 5 \text{ then } T_1 \text{ else } T_2$)
- Indexed session types: $\text{countDown} :: \Pi x: \text{Nat}. \text{stype}$
- Ability to specify and prove properties of processes within the theory:

  \[
  \text{simpleCounter} :: \text{Nat} \rightarrow \{ \text{simpleCounter} \} \\
  \text{simpleCounter} = \lambda x: \text{Nat}. \{ \ldots \}
  \]
Richer Type Theories
Where are we?

Monadic Dependencies (Ongoing work)
By adding the (contextual) monad to a dependent type theory we get:

- Relations on processes as type families (e.g. \( \Pi x: \{ z:A \} . \Pi y: \{ z:B \} . \text{type} \))
- Value dependent communication (e.g. \( \forall x: \text{Nat}. \text{if } x = 5 \text{ then } T_1 \text{ else } T_2 \))
- Indexed session types: \( \text{countDown} :: \Pi x: \text{Nat}. \text{stype} \)
- Ability to specify and prove properties of processes within the theory:

  \[
  \begin{align*}
  \text{simpleCounter} & :: \text{Nat} \to \{ \text{simpleCounter} \} \\
  \text{simpleCounter} & = \lambda x: \text{Nat}. \{ \ldots \} \\
  \text{corrCount} & :: \Pi x: \text{Nat}. \Pi y: \{ \text{simpleCounter} \} . \text{type} \\
  \ldots
  \end{align*}
  \]
Richer Type Theories
Where are we?

Monadic Dependencies (Ongoing work)

By adding the (contextual) monad to a dependent type theory we get:

- Relations on processes as type families (e.g. \( \prod x: \{ z:A \}. \prod y: \{ z:B \}. \text{type} \))
- Value dependent communication (e.g. \( \forall x: \text{Nat}. \text{if } x = 5 \text{ then } T_1 \text{ else } T_2 \))
- Indexed session types: countDown :: \( \prod x: \text{Nat}. \text{stype} \)
- Ability to specify and prove properties of processes within the theory:

\[
\begin{align*}
\text{simpleCounter} & :: \ \text{Nat} \rightarrow \{ \text{simpleCounter} \} \\
\text{simpleCounter} & = \lambda x: \text{Nat}. \{ \ldots \} \\
\text{corrCount} & :: \ \prod x: \text{Nat}. \prod y: \{ \text{simpleCounter} \}. \text{type} \\
\ldots & \\
\text{correctness} & :: \ \prod n: \text{Nat}. \text{corrCount } n (\text{simpleCounter}(n))
\end{align*}
\]
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?
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.
Proof Conversions

Commuting Conversions

Commuting Conversions induce a congruence \( \simeq_c \) on typed processes

\( \otimes L/\otimes L \) Commuting Conversion

\[
x : A \otimes B, z : C \otimes D \implies x(y).z(w).P \simeq_c z(w).x(y).P :: v : E
\]

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

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.
## Proof Conversions
Typed Contextual Equivalence

### Typed Contextual Equivalence
- How to define this equivalence in a tractable way?
- Typed Contextual **Bisimilarity**.
Proof Conversions
Typed Contextual Equivalence

Typed Contextual Equivalence
- How to define this equivalence in a tractable way?
- Typed Contextual Bisimilarity.

Contextual Equivalence
- Contextual: For all typed contexts...
- Typed bisimilarity on closed processes:
  - \( P \sim Q :: x:A \rightarrow B \) iff \( P \xrightarrow{x(y)} P' \) implies \( Q \xrightarrow{x(y)} Q' \) and \( \forall R. \Rightarrow \Rightarrow R :: y:A \) we have \((\nu y)(P | R) \sim (\nu y)(Q | R) :: x:B\)
  - \( P \sim Q :: x:C \) iff \( P \xrightarrow{T} P' \) implies \( Q \Rightarrow Q' \) and \( P' \sim Q' :: x:C\).
  - ...

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

- Suppose input along \( x \) matches an output in \( R_x \) on the left proc.
Proof Conversions

Issue

⊗L/⊗L Conversion Revisited

\((νx)(νz)(x(y).z(w).P | R_x | S_z) \sim (νx)(νz)(z(w).x(y).P | R_x | S_z) :: ν:E?\)

- 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 \(z\)?
Proof Conversions

Issue

⊗L/⊗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!
Proof Conversions

Issue

⊗L/⊗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 Equivalence

- Can we develop a uniform solution?
- Inspiration from functional “world”: Linear Logical Relations!
Proof Conversions
Termination and Equivalence

Linear Logical Relations [Pérez et al. 12]

- Termination: Inductively defined unary predicate.
- Contextual Equivalence: Extension to relational setting.
Proof Conversions
Termination and Equivalence

Linear Logical Relations [Pérez et al. 12]
- Termination: Inductively defined unary predicate.
- Contextual Equivalence: Extension to relational setting.

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 \( \forall R \in \mathcal{L}[y:A].(\nu y)(R \mid P) \in \mathcal{L}[\Gamma; \Delta \vdash T] \).
Proof Conversions
Termination and Equivalence

Linear Logical Relations [Pérez et al. 12]

- Termination: Inductively defined unary predicate.
- Contextual Equivalence: Extension to relational setting.

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 \( \forall R \in \mathcal{L}[y:A].(\nu y)(R \parallel P) \in \mathcal{L}[\Gamma; \Delta \vdash T] \).

- Base case is inductive on types:
  
  \( P \in \mathcal{L}[z:A \rightarrow B] \triangleq \) if \( P \xrightarrow{z(y)} P' \) then \( \forall Q \in \mathcal{L}[y:A].(\nu y)(P' \parallel Q) \in \mathcal{L}[z:B] \)
  
  ...
**Proof Conversions**

**Termination and Equivalence**

**Linear Logical Relations [Pérez et al. 12]**
- Termination: Inductively defined unary predicate.
- Contextual Equivalence: Extension to relational setting.

**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:
  - $P \in \mathcal{L}[z:A \to B] \triangleq P \xrightarrow{z(y)} P'$ then $\forall Q \in \mathcal{L}[y:A].(\nu y)(P' \mid Q) \in \mathcal{L}[z:B]$
  - ...

**Typing implies Termination**
If $\Gamma; \Delta \vdash P :: T$ then $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$. 
Typed Equivalence

- 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 \( \forall R. \vdash R :: y:A, \Gamma; \Delta \vdash (\nu y)(P|R)\mathcal{R}(\nu y)(Q|R) :: T \).
Proof Conversions
Termination and Equivalence

Typed Equivalence

- Relational generalization of the predicate.
- Inductive on typing derivations: \( \Gamma; \Delta \vdash P \triangleleft Q :: T \)
  - If \( \Gamma; \Delta, y:A \vdash P \triangleleft Q :: T \) then \( \forall R. \vdash R :: y:A, \Gamma; \Delta \vdash (\nu y)(P \mid R) \triangleleft (\nu y)(Q \mid R) :: T \).
- Base case is inductive on types:
  - \( \vdash P \triangleleft Q :: x:A \rightarrow B \) iff \( P \xrightarrow{x(y)} P' \) implies \( Q \xrightarrow{x(y)} Q' \) and \( \forall R. \vdash R :: y:A \) we have \( \vdash (\nu y)(P \mid R) \triangleleft (\nu y)(Q \mid R) :: x:B \)
  - ...
Typed Equivalence

- Relational generalization of the predicate.
- Inductive on typing derivations: $\Gamma; \Delta \vdash P \Rightarrow Q :: T$
  - If $\Gamma; \Delta, y:A \vdash P \Rightarrow Q :: T$ then $\forall R. \vdash R :: y:A, \Gamma; \Delta \vdash (\nu y)(P|R) \Rightarrow (\nu y)(Q|R) :: T$.
- Base case is inductive on types:
  - $\vdash P \Rightarrow Q :: x:A \rightarrow B$ iff $P \xrightarrow{x(y)} P'$ implies $Q \xrightarrow{x(y)} Q'$ and $\forall R. \vdash R :: y:A$ we have $\vdash (\nu y)(P|R) \Rightarrow (\nu y)(Q|R) :: x:B$
  - ...

Soundness of Commuting Conversions

If $\Gamma; \Delta \vdash P \simeq c Q :: T$ then $\Gamma; \Delta \vdash P \simeq Q :: T$
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.

Validating Isomorphisms
If \(A \simeq B\) then \(A \simeq_S B\).
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 $y:B \vdash Q :: x:A$. 

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

Bernardo Toninho mrg.doc.ic.ac.uk 29/32
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 \(y: B \vdash Q :: x: A\).
- \(x: A \vdash (\nu y)(P \mid Q) \approx [x \leftrightarrow z] :: z: A\).
- \(y: B \vdash (\nu x)(Q \mid P) \approx [y \leftrightarrow z] :: z: B\).
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 $y:B \vdash Q :: x:A$.
- $x:A \vdash (\nu y)(P \mid Q) \approx [x \leftrightarrow z] :: z:A$.
- $y:B \vdash (\nu x)(Q \mid P) \approx [y \leftrightarrow z] :: z:B$.

Validating Isomorphisms
If $A \simeq B$ then $A \simeq_S B$. 
Summary

So far:

- 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.

Not in this talk:

- Asynchrony [DeYoung12]
- Parametricity [Caires et al12]
- Inductive and Coinductive Session Types [Toninho et al14,LindleyMorris16]
- Monitoring and blame [Jia et al16]
- Shared resources and non-determinism [Atkey et al16,BalzerPfenning17]
- Multiparty sessions [Carbone et al15,16]
Thank you!
Questions?
Session Types and Linear Logic

Bernardo Toninho and Nobuko Yoshida

University of Bath,
November 28, 2017