Depending on Session-Typed Processes

Mathematical Foundations of Programming Semantics

Bernardo Toninho, Nobuko Yoshida

June 7, 2018

Background

Session Types

Message-Passing 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."

Background

Session Types

Message-Passing 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 Types [Honda93]

- Types are descriptions of communication behaviour, assigned to channels.
- A way of guaranteeing communication discipline, statically.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select.
- Duality ensures progress (deadlock-freedom).

Background

Linear Logic and Session Types

Linear Logic [Girard87]

- A marriage of classical dualities and constructivism.
- A logic of resources and interaction.
- Resource independence captures some non-determinism/concurrency.

Session Types and LL [CairesPfenning12]

- Its possible to interpret session types as linear logic propositions.
- Linear logic proofs as process typing derivations.
- Proof dynamics as process dynamics.
- Deadlock-freedom by typing (stronger).

Background

Linear Logic and Session Types

Propositional Linear Logic as Session Types

Types are assigned to channels:

- Linear Implication ($A \multimap B$): Receive a channel of type A and continue with B.

Background

Linear Logic and Session Types

Propositional Linear Logic as Session Types

Types are assigned to channels:

- Linear Implication ($A \multimap B$): Receive a channel of type A and continue with B.
- Multiplicative Conjunction ($A \otimes B$): Send a channel of type A and continue as B.

Background

Linear Logic and Session Types

Propositional Linear Logic as Session Types

Types are assigned to channels:

- Linear Implication ($A \multimap B$): Receive a channel of type A and continue with B.
- Multiplicative Conjunction ($A \otimes B$): Send a channel of type A and continue as B.
- Additive Conjunction ($A \,\&\, B$): Receive either inl and continue as A or inr and continue as B.
- ...

Background

Linear Logic and Session Types

Propositional Linear Logic as Session Types

Types are assigned to channels:

- Linear Implication ($A \multimap B$): Receive a channel of type A and continue with B.
- Multiplicative Conjunction ($A \otimes B$): Send a channel of type A and continue as B.
- Additive Conjunction ($A \,\&\, B$): Receive either inl and continue as A or inr and continue as B.
- ...

Limitations

- Only express simple communication patterns.
- No interesting properties of exchanged data.

Background

Addressing Limitations

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

- Correspondence with 1st-order LL (parametric in the domain of quantification τ):
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.

Background

Addressing Limitations

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

- Correspondence with 1st-order LL (parametric in the domain of quantification τ):
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.
- Properties/assertions on data, statically (e.g. $\forall x{:}\mathrm{Nat}.\forall y{:}(x > 5).A$).
- Related extensions on type refinements, type assertions, etc.

Background

Addressing Limitations

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

- Correspondence with 1st-order LL (parametric in the domain of quantification τ):
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.
- Properties/assertions on data, statically (e.g. $\forall x{:}\mathsf{Nat}.\forall y{:}(x > 5).A$).
- Related extensions on type refinements, type assertions, etc.

Limitations

- No way of having protocol structure depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".

Motivation

Dependent Session Types

We want to be able to do these things!

Motivation

Dependent Session Types

We want to be able to do these things!

Goals

- Enrich the type structure to allow for truly data dependent protocols.
- Allowing for reasoning about processes, mirroring functional type theories.

Motivation

Dependent Session Types

We want to be able to do these things!

Goals

- Enrich the type structure to allow for truly data dependent protocols.
- Allowing for reasoning about processes, mirroring functional type theories.
- Desiderata:
  - Functions can depend on processes and vice-versa.
  - Decidable type checking.
  - Faithfully encode dependent functions with processes.

Motivation

Dependent Session Types

We want to be able to do these things!

Goals

- Enrich the type structure to allow for truly data dependent protocols.
- Allowing for reasoning about processes, mirroring functional type theories.
- Desiderata:
  - Functions can depend on processes and vice-versa.
  - Decidable type checking.
  - Faithfully encode dependent functions with processes.

Contributions

- A type theory of (dependent) functions and processes.
- Type safety of the theory.
- An encoding of dependent functions with processes.

Outline

- Depending on Session-Typed Processes
  - Functions and Processes
  - Typing
  - Definitional Equality
  - Examples

Outline

- Depending on Session-Typed Processes
  - Functions and Processes
  - Typing
  - Definitional Equality
  - Examples
- Encoding Dependent Functions with Processes
  - Encoding
  - Properties

Depending on Processes

Data-Dependent Protocol

Boolean-driven Communication

Depending on Processes

Data-Dependent Protocol

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B\ =\ A \qquad \texttt{if}\ \mathsf{false}\ A\ B\ =\ B$$

Depending on Processes

Data-Dependent Protocol

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B\ =\ A \qquad \texttt{if}\ \mathsf{false}\ A\ B\ =\ B$$

$$T \triangleq \forall x{:}\mathsf{Bool}.\texttt{if}\ x\ (\mathsf{Nat} \wedge \mathbf{1})\ (\mathsf{Bool} \wedge \mathbf{1})$$

Depending on Processes

Data-Dependent Protocol

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B\ =\ A \qquad \texttt{if}\ \mathsf{false}\ A\ B\ =\ B$$

$$T \triangleq \forall x{:}\mathsf{Bool}.\texttt{if}\ x\ (\mathsf{Nat} \wedge \mathbf{1})\ (\mathsf{Bool} \wedge \mathbf{1})$$

Depending on Processes

Data-Dependent Protocol

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B\ =\ A \qquad \texttt{if}\ \mathsf{false}\ A\ B\ =\ B$$

$$T \triangleq \forall x{:}\mathsf{Bool}.\texttt{if}\ x\ (\mathsf{Nat} \wedge \mathbf{1})\ (\mathsf{Bool} \wedge \mathbf{1})$$

Depending on Processes

Data-Dependent Protocol

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B\ =\ A \qquad \texttt{if}\ \mathsf{false}\ A\ B\ =\ B$$

$$T \triangleq \forall x{:}\mathsf{Bool}.\texttt{if}\ x\ (\mathsf{Nat} \wedge \mathbf{1})\ (\mathsf{Bool} \wedge \mathbf{1})$$

$$\vdash z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{true} \Rightarrow z\langle 23\rangle.\mathbf{0},\ \mathsf{false} \Rightarrow z\langle \mathsf{true}\rangle.\mathbf{0}) :: z{:}T$$

Depending on Processes

Data-Dependent Protocol

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B\ =\ A \qquad \texttt{if}\ \mathsf{false}\ A\ B\ =\ B$$

$$T \triangleq \forall x{:}\mathsf{Bool}.\texttt{if}\ x\ (\mathsf{Nat} \wedge \mathbf{1})\ (\mathsf{Bool} \wedge \mathbf{1})$$

$$\vdash z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{true} \Rightarrow z\langle 23\rangle.\mathbf{0},\ \mathsf{false} \Rightarrow z\langle \mathsf{true}\rangle.\mathbf{0}) :: z{:}T$$

$$\nvdash z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{false} \Rightarrow z\langle 23\rangle.\mathbf{0},\ \mathsf{true} \Rightarrow z\langle \mathsf{true}\rangle.\mathbf{0}) :: z{:}T$$

Depending on Processes

A Language of Functions and Processes

Kinds $\quad K, K' \quad ::= \quad \mathsf{type} \mid \mathsf{stype}$

Depending on Processes

A Language of Functions and Processes

Kinds $K, K' ::= \text{type} \mid \text{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\}$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\ B$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\ B$
Terms	M, N	$::=$	$\lambda x{:}\tau.M \mid M\ N \mid x$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\ B$
Terms	M, N	$::=$	$\lambda x{:}\tau.M \mid M\ N \mid x \mid \{c \leftarrow P \leftarrow \overline{d_i}\}$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\ B$
Terms	M, N	$::=$	$\lambda x{:}\tau.M \mid M\ N \mid x \mid \{c \leftarrow P \leftarrow \overline{d_i}\}$
Processes	P, Q	$::=$	$\overline{c}\langle d\rangle.P \mid c\langle M\rangle.P \mid c(x).P$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\ B$
Terms	M, N	$::=$	$\lambda x{:}\tau.M \mid M\ N \mid x \mid \{c \leftarrow P \leftarrow \overline{d_i}\}$
Processes	P, Q	$::=$	$\overline{c}\langle d\rangle.P \mid c\langle M\rangle.P \mid c(x).P$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\ B$
Terms	M, N	$::=$	$\lambda x{:}\tau.M \mid M\ N \mid x \mid \{c \leftarrow P \leftarrow \overline{d_i}\}$
Processes	P, Q	$::=$	$\overline{c}\langle d\rangle.P \mid c\langle M\rangle.P \mid c(x).P \mid c.\mathsf{case}\{\overline{l_i \Rightarrow P_i}\} \mid c.l; P$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\ B$
Terms	M, N	$::=$	$\lambda x{:}\tau.M \mid M\ N \mid x \mid \{c \leftarrow P \leftarrow \overline{d_i}\}$
Processes	P, Q	$::=$	$\overline{c}\langle d\rangle.P \mid c\langle M\rangle.P \mid c(x).P \mid c.\mathsf{case}\{\overline{l_i \Rightarrow P_i}\} \mid c.l; P$
		$\mid$	$c \leftarrow M \leftarrow \overline{d_i}; Q$

Depending on Processes

A Language of Functions and Processes

Kinds	K, K'	$::=$	$\mathsf{type} \mid \mathsf{stype} \mid \Pi x{:}\tau.K \mid \Pi t{:}K.K'$
Types	τ, σ	$::=$	$\Pi x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \lambda t :: K.\tau \mid \tau\ \sigma \mid \{\overline{d_i{:}B_i} \vdash c{:}A\}$
Sessions	A, B	$::=$	$A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1}$
		$\mid$	$\&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\ B$
Terms	M, N	$::=$	$\lambda x{:}\tau.M \mid M\ N \mid x \mid \{c \leftarrow P \leftarrow \overline{d_i}\}$
Processes	P, Q	$::=$	$\overline{c}\langle d\rangle.P \mid c\langle M\rangle.P \mid c(x).P \mid c.\mathsf{case}\{\overline{l_i \Rightarrow P_i}\} \mid c.l; P$
		$\mid$	$c \leftarrow M \leftarrow \overline{d_i}; Q \mid (\nu c)(P \mid Q) \mid [c \leftrightarrow d] \mid \mathbf{0}$

- Session typing can depend on sent and received functional terms.
- Functions can construct processes through the monad.
- Dependencies between both "layers" + type-level functions.

Depending on Processes

Typing

Typing Judgements

$$\overbrace{x_1{:}\tau_1, \ldots, x_m{:}\tau_m}^{\Psi}; \overbrace{y_1{:}A_1, \ldots, y_n{:}A_n}^{\Delta} \vdash P :: z{:}A$$

P offers A along z if composed with sessions in Δ, with free (term) variables in Ψ.

Depending on Processes

Typing

Typing Judgements

$$\Psi \vdash M : \tau$$

M has type τ, with free variables recorded in Ψ

Depending on Processes

Typing

Typing Judgements

$$\overbrace{x_1{:}\tau_1, \ldots, x_m{:}\tau_m}^{\Psi}; \overbrace{y_1{:}A_1, \ldots, y_n{:}A_n}^{\Delta} \vdash P :: z{:}A$$

P offers A along z if composed with sessions in Δ, with free (term) variables in Ψ.

Kind and Equality Judgements

$\Psi \vdash \tau :: K$	τ is a (functional) type of kind K in context Ψ.
$\Psi \vdash A :: K$	A is a session type of kind K in context Ψ.

Depending on Processes

Typing

Typing Judgements

$$\overbrace{x_1{:}\tau_1, \ldots, x_m{:}\tau_m}^{\Psi}; \overbrace{y_1{:}A_1, \ldots, y_n{:}A_n}^{\Delta} \vdash P :: z{:}A$$

P offers A along z if composed with sessions in Δ, with free (term) variables in Ψ.

Kind and Equality Judgements

$\Psi \vdash \tau :: K$	τ is a (functional) type of kind K in context Ψ.
$\Psi \vdash A :: K$	A is a session type of kind K in context Ψ.
$\Psi \vdash \tau = \sigma :: K$	Types τ and σ are equal of kind K.
$\Psi \vdash A = B :: K$	Session types A and B are equal of kind K.

Depending on Processes

Typing

Typing Judgements

$$\overbrace{x_1{:}\tau_1, \ldots, x_m{:}\tau_m}^{\Psi}; \overbrace{y_1{:}A_1, \ldots, y_n{:}A_n}^{\Delta} \vdash P :: z{:}A$$

P offers A along z if composed with sessions in Δ, with free (term) variables in Ψ.

Kind and Equality Judgements

$\Psi \vdash \tau :: K$	τ is a (functional) type of kind K in context Ψ.
$\Psi \vdash A :: K$	A is a session type of kind K in context Ψ.
$\Psi \vdash \tau = \sigma :: K$	Types τ and σ are equal of kind K.
$\Psi \vdash A = B :: K$	Session types A and B are equal of kind K.
$\Psi \vdash M = N : \tau$	Terms M and N are equal of type τ.
$\Psi; \Delta \vdash P = Q :: z{:}A$	Processes P and Q are equal with typing $z{:}A$.

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}I)\ \frac{\Psi; \overline{d_i{:}A_i} \vdash P :: c{:}A}{\Psi \vdash \{c \leftarrow P \leftarrow \overline{d_i}\} : \{\overline{d_i : A_i} \vdash c{:}A\}}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}I)\ \frac{\Psi; \overline{d_i{:}A_i} \vdash P :: c{:}A}{\Psi \vdash \{c \leftarrow P \leftarrow \overline{d_i}\} : \{\overline{d_i : A_i} \vdash c{:}A\}}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}E)\ \frac{\Delta' = \overline{d_i : B_i} \quad \Psi \vdash M : \{\overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{d_i}; Q :: z{:}C}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}E)\ \frac{\Delta' = \overline{d_i : B_i} \quad \Psi \vdash M : \{\overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{d_i}; Q :: z{:}C}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}E)\ \frac{\Delta' = \overline{d_i : B_i} \quad \Psi \vdash M : \{\overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{d_i}; Q :: z{:}C}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}E)\ \frac{\Delta' = \overline{d_i : B_i} \quad \Psi \vdash M : \{\overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{d_i}; Q :: z{:}C}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}E)\ \frac{\Delta' = \overline{d_i : B_i} \quad \Psi \vdash M : \{\overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{d_i}; Q :: z{:}C}$$

Dependent I/O

$$(\forall R)\ \frac{\Psi \vdash \tau :: \text{type} \quad \Psi, x{:}\tau\ ; \Delta \vdash P :: c{:}A}{\Psi; \Delta \vdash c(x{:}\tau).P :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}E)\ \frac{\Delta' = \overline{d_i : B_i} \quad \Psi \vdash M : \{\overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{d_i}; Q :: z{:}C}$$

Dependent I/O

$$(\forall R)\ \frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau\ ; \Delta \vdash P :: c{:}A}{\Psi; \Delta \vdash c(x{:}\tau).P :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}E)\ \frac{\Delta' = \overline{d_i : B_i} \quad \Psi \vdash M : \{\overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{d_i}; Q :: z{:}C}$$

Dependent I/O

$$(\forall R)\ \frac{\Psi \vdash \tau :: \text{type} \quad \Psi, x{:}\tau\ ; \Delta \vdash P :: c{:}A}{\Psi; \Delta \vdash c(x{:}\tau).P :: c{:}\forall x{:}\tau.A} \qquad (\forall L)\ \frac{\Psi \vdash M{:}\tau \quad \Psi; \Delta, c{:}A\{M/x\} \vdash Q :: d{:}D}{\Psi; \Delta, c{:}\forall x{:}\tau.A \vdash c\langle M\rangle_{\forall x:\tau.A}.Q :: d{:}D}$$

Depending on Processes

Typing

Typing Rules – Monad

$$(\{\}E)\ \frac{\Delta' = \overline{d_i : B_i} \quad \Psi \vdash M : \{\overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{d_i}; Q :: z{:}C}$$

Dependent I/O

$$(\forall R)\ \frac{\Psi \vdash \tau :: \text{type} \quad \Psi, x{:}\tau\ ; \Delta \vdash P :: c{:}A}{\Psi; \Delta \vdash c(x{:}\tau).P :: c{:}\forall x{:}\tau.A} \qquad (\forall L)\ \frac{\Psi \vdash M{:}\tau \quad \Psi; \Delta, c{:}A\{M/x\} \vdash Q :: d{:}D}{\Psi; \Delta, c{:}\forall x{:}\tau.A \vdash c\langle M\rangle_{\forall x:\tau.A}.Q :: d{:}D}$$

Depending on Processes

Typing and Equality

Conversion

$$\frac{\Psi \vdash M : \tau \quad \Psi \vdash \tau = \sigma :: \text{type}}{\Psi \vdash M : \sigma}\ (\text{Conv}) \qquad \frac{\Psi; \Delta \vdash P :: z{:}A \quad \Psi \vdash A = B :: \text{stype}}{\Psi; \Delta \vdash P :: z{:}B}\ (\text{ConvR})$$

Depending on Processes

Typing and Equality

Conversion

$$\frac{\Psi \vdash M : \tau \quad \Psi \vdash \tau = \sigma :: \text{type}}{\Psi \vdash M : \sigma}\ (\text{Conv}) \qquad \frac{\Psi; \Delta \vdash P :: z{:}A \quad \Psi \vdash A = B :: \text{stype}}{\Psi; \Delta \vdash P :: z{:}B}\ (\text{ConvR})$$

- What is the "right" notion of process equality?

Depending on Processes

Typing and Equality

Conversion

$$\frac{\Psi \vdash M : \tau \quad \Psi \vdash \tau = \sigma :: \text{type}}{\Psi \vdash M : \sigma}\ (\text{Conv}) \qquad \frac{\Psi; \Delta \vdash P :: z{:}A \quad \Psi \vdash A = B :: \text{stype}}{\Psi; \Delta \vdash P :: z{:}B}\ (\text{ConvR})$$

- What is the "right" notion of process equality?

$$\text{sendOne} \quad = \quad \{c \leftarrow c\langle 1 \rangle.\mathbf{0}\}$$

Depending on Processes

Typing and Equality

Conversion

$$\frac{\Psi \vdash M : \tau \quad \Psi \vdash \tau = \sigma :: \text{type}}{\Psi \vdash M : \sigma}\ (\text{Conv}) \qquad \frac{\Psi; \Delta \vdash P :: z{:}A \quad \Psi \vdash A = B :: \text{stype}}{\Psi; \Delta \vdash P :: z{:}B}\ (\text{ConvR})$$

- What is the "right" notion of process equality?

$$\begin{aligned} \text{sendOne} &= \{c \leftarrow c\langle 1\rangle.\mathbf{0}\} \\ \text{sendOne}' &= \{c \leftarrow (\boldsymbol{\nu} d)(d\langle 1\rangle.\mathbf{0} \mid d(x).c\langle x\rangle.\mathbf{0})\} \end{aligned}$$

Depending on Processes

Typing and Equality

Conversion

$$\frac{\Psi \vdash M : \tau \quad \Psi \vdash \tau = \sigma :: \text{type}}{\Psi \vdash M : \sigma}\ (\text{Conv}) \qquad \frac{\Psi; \Delta \vdash P :: z{:}A \quad \Psi \vdash A = B :: \text{stype}}{\Psi; \Delta \vdash P :: z{:}B}\ (\text{ConvR})$$

- What is the "right" notion of process equality?

$$\begin{aligned} \text{sendOne} &= \{c \leftarrow c\langle 1\rangle.\mathbf{0}\} \\ \text{sendOne}' &= \{c \leftarrow (\nu d)(d\langle 1\rangle.\mathbf{0} \mid d(x).c\langle x\rangle.\mathbf{0})\} \end{aligned}$$

$$(\nu d)(d\langle 1\rangle.\mathbf{0} \mid d(x).c\langle x\rangle.\mathbf{0}) \rightarrow c\langle 1\rangle.\mathbf{0}$$

Depending on Processes

Typing and Equality

Conversion

$$\frac{\Psi \vdash M : \tau \quad \Psi \vdash \tau = \sigma :: \text{type}}{\Psi \vdash M : \sigma}\ (\text{Conv}) \qquad \frac{\Psi; \Delta \vdash P :: z{:}A \quad \Psi \vdash A = B :: \text{stype}}{\Psi; \Delta \vdash P :: z{:}B}\ (\text{ConvR})$$

- What is the "right" notion of process equality?

$$\begin{aligned} \text{sendOne} &= \{c \leftarrow c\langle 1\rangle.\mathbf{0}\} \\ \text{sendOne}' &= \{c \leftarrow (\nu d)(d\langle 1\rangle.\mathbf{0} \mid d(x).c\langle x\rangle.\mathbf{0})\} \end{aligned}$$

$$(\nu d)(d\langle 1\rangle.\mathbf{0} \mid d(x).c\langle x\rangle.\mathbf{0}) \rightarrow c\langle 1\rangle.\mathbf{0}$$

$$\vdash \text{sendOne} = \text{sendOne}' : \{\exists x{:}\text{Nat}.\mathbf{1}\}$$

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction?

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction?

Our litmus test: Can we internally encode λ-terms with "just" processes?

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- If we have η in functions, we need "η-like" expansion principles for processes:

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- If we have η in functions, we need "η-like" expansion principles for processes:

$$(\text{PEq}\forall\eta)\ \frac{}{\Psi;\, d{:}\forall x{:}\tau.A \vdash c(x).d\langle x\rangle.[d \leftrightarrow c]_A = [d \leftrightarrow c]_{\forall x:\tau.A} :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- If we have η in functions, we need "η-like" expansion principles for processes:

$$(\text{PEq}\forall\eta)\ \frac{}{\Psi;\, d{:}\forall x{:}\tau.A \vdash c(x).d\langle x\rangle.[d \leftrightarrow c]_A = [d \leftrightarrow c]_{\forall x:\tau.A} :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- If we have η in functions, we need "η-like" expansion principles for processes:

$$(\text{PEq}\forall\eta)\ \frac{}{\Psi;\, d{:}\forall x{:}\tau.A \vdash c(x).d\langle x\rangle.[d \leftrightarrow c]_A = [d \leftrightarrow c]_{\forall x:\tau.A} :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- If we have η in functions, we need "η-like" expansion principles for processes:

$$(\text{PEq}\forall\eta)\ \frac{}{\Psi;\, d{:}\forall x{:}\tau.A \vdash c(x).d\langle x\rangle.[d \leftrightarrow c]_A = [d \leftrightarrow c]_{\forall x:\tau.A} :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction + η principles?

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction + η principles?

Our litmus test: Can we internally encode λ-terms with "just" processes?

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction + η principles?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- Turns out η in λ requires a bit more from processes...

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction + η principles?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- Turns out η in λ requires a bit more from processes...
- We need commutting conversions:

$$(\text{PEqCC}\forall)\ \frac{\Psi; \Delta \vdash P :: d{:}B \quad \Psi, x{:}\tau; \Delta', d{:}B \vdash Q :: c{:}A}{\Psi; \Delta, \Delta' \vdash (\nu d)(P \mid c(x).Q) = c(x).(\nu d)(P \mid Q) :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction + η principles?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- Turns out η in λ requires a bit more from processes...
- We need commutting conversions:

$$(\text{PEqCC}\forall)\ \frac{\Psi; \Delta \vdash P :: d{:}B \quad \Psi, x{:}\tau; \Delta', d{:}B \vdash Q :: c{:}A}{\Psi; \Delta, \Delta' \vdash (\nu d)(P \mid c(x).Q) = c(x).(\nu d)(P \mid Q) :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Equality

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction + η principles?

Our litmus test: Can we internally encode λ-terms with "just" processes?

- Turns out η in λ requires a bit more from processes...
- We need commutting conversions:

$$(\text{PEqCC}\forall)\ \frac{\Psi;\Delta \vdash P :: d{:}B \quad \Psi, x{:}\tau;\Delta', d{:}B \vdash Q :: c{:}A}{\Psi;\Delta,\Delta' \vdash (\nu d)(P \mid c(x).Q) = c(x).(\nu d)(P \mid Q) :: c{:}\forall x{:}\tau.A}$$

Depending on Processes

Metatheory

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction + η principles + CC.

Depending on Processes

Metatheory

Equality

- For λ-terms, full $\beta\eta$-conversion (usual).
- For processes, full (i.e. under binders/prefixes) reduction + η principles + CC.

Metatheorical Properties

In a nutshell:

- Unicity of Types/Session Types.
- Subject Reduction for Terms and Processes.
- Progress for terms / Deadlock-freedom for Processes.

Depending on Processes

Example – Counting Down

Simply Counting Down

Depending on Processes

Example – Counting Down

Simply Counting Down

$$\begin{array}{lcl} \mathsf{simpleCounterT} & :: & \mathsf{stype} \\ \mathsf{simpleCounterT} & = & \oplus\{\mathsf{dec} : \mathsf{Nat} \wedge \mathsf{simpleCounterT}, \mathsf{done} : \mathbf{1}\} \end{array}$$

Depending on Processes

Example – Counting Down

Simply Counting Down

$$
\begin{array}{lcl}
\mathsf{simpleCounterT} & :: & \mathsf{stype} \\
\mathsf{simpleCounterT} & = & \oplus\{\mathsf{dec} : \mathsf{Nat} \wedge \mathsf{simpleCounterT}, \mathsf{done} : \mathbf{1}\} \\
\mathsf{simpleCounter}\ n & = & \{\ldots\}
\end{array}
$$

Depending on Processes

Example – Counting Down

Simply Counting Down

$$
\begin{array}{lcl}
\mathsf{simpleCounterT} & :: & \mathsf{stype} \\
\mathsf{simpleCounterT} & = & \oplus\{\mathsf{dec} : \mathsf{Nat} \wedge \mathsf{simpleCounterT}, \mathsf{done} : \mathbf{1}\} \\
\mathsf{simpleCounter}\ n & = & \{\ldots\} \\
\mathsf{corrCount} & :: & \Pi x{:}\mathsf{Nat}.\Pi y{:}\{\mathsf{simpleCounterT}\}.\mathsf{type}
\end{array}
$$

Depending on Processes

Example – Counting Down

Simply Counting Down

$$
\begin{array}{lcl}
\mathsf{simpleCounterT} & :: & \mathsf{stype} \\
\mathsf{simpleCounterT} & = & \oplus\{\mathsf{dec} : \mathsf{Nat} \wedge \mathsf{simpleCounterT}, \mathsf{done} : \mathbf{1}\} \\
\mathsf{simpleCounter}\ n & = & \{\ldots\} \\
\mathsf{corrCount} & :: & \Pi x{:}\mathsf{Nat}.\Pi y{:}\{\mathsf{simpleCounterT}\}.\mathsf{type} \\
\mathsf{corr}_z & : & \mathsf{corrCount}\ \mathsf{z}\ \{c \leftarrow c.\mathsf{done}; \mathbf{0}\}
\end{array}
$$

Depending on Processes

Example – Counting Down

Simply Counting Down

$$
\begin{array}{lcl}
\mathsf{simpleCounterT} & :: & \mathsf{stype} \\
\mathsf{simpleCounterT} & = & \oplus\{\mathsf{dec} : \mathsf{Nat} \wedge \mathsf{simpleCounterT}, \mathsf{done} : \mathbf{1}\} \\
\mathsf{simpleCounter}\ n & = & \{\ldots\} \\
\mathsf{corrCount} & :: & \Pi x{:}\mathsf{Nat}.\Pi y{:}\{\mathsf{simpleCounterT}\}.\mathsf{type} \\
\mathsf{corr}_z & : & \mathsf{corrCount}\ \mathsf{z}\ \{c \leftarrow c.\mathsf{done}; \mathbf{0}\} \\
\mathsf{corr}_n & : & \Pi n{:}\mathsf{Nat}.\Pi P{:}\{\mathsf{simpleCounterT}\}.\mathsf{corrCount}\ n\ P \rightarrow \\
 & & \mathsf{corrCount}\ (\mathsf{succ}(n))\ \{c \leftarrow c.\mathsf{dec}; c\langle\mathsf{succ}(n)\rangle.c \leftarrow P\}
\end{array}
$$

Depending on Processes

Example – Counting Down

Simply Counting Down

$$
\begin{array}{lcl}
\mathsf{simpleCounterT} & :: & \mathsf{stype} \\
\mathsf{simpleCounterT} & = & \oplus\{\mathsf{dec} : \mathsf{Nat} \wedge \mathsf{simpleCounterT}, \mathsf{done} : \mathbf{1}\} \\
\mathsf{simpleCounter}\ n & = & \{\ldots\} \\
\mathsf{corrCount} & :: & \Pi x{:}\mathsf{Nat}.\Pi y{:}\{\mathsf{simpleCounterT}\}.\mathsf{type} \\
\mathsf{corr}_z & : & \mathsf{corrCount}\ \mathsf{z}\ \{c \leftarrow c.\mathsf{done}; \mathbf{0}\} \\
\mathsf{corr}_n & : & \Pi n{:}\mathsf{Nat}.\Pi P{:}\{\mathsf{simpleCounterT}\}.\mathsf{corrCount}\ n\ P \rightarrow \\
 & & \mathsf{corrCount}\ (\mathsf{succ}(n))\ \{c \leftarrow c.\mathsf{dec}; c\langle \mathsf{succ}(n)\rangle.c \leftarrow P\} \\
\mathsf{prf} & : & \Pi n{:}\mathsf{Nat}.\mathsf{corrCount}\ n\ (\mathsf{simpleCounter}(n)) \\
\mathsf{prf}\ \ \mathsf{z} & = & \mathsf{corr}_z \\
\mathsf{prf}\ \ (\mathsf{succ}(n)) & = & \mathsf{corr}_n\ n\ (\mathsf{simpleCounter}(n))\ (\mathsf{prf}\ n)
\end{array}
$$

Encoding

Definition

Validating our system by encoding functions in the process layer:

- Encoding of a term M is indexed by a result channel z – $[\![M]\!]_z$.

Encoding

Definition

Validating our system by encoding functions in the process layer:

- Encoding of a term M is indexed by a result channel z – $[\![M]\!]_z$.
- Target calculus is a higher-order (i.e. process-passing) process calculus:

$$
\begin{array}{llll}
\text{Types} & \tau, \sigma & ::= & \{\overline{d_i{:}B_i} \vdash c{:}A\} \\
\text{Sessions} & A, B & ::= & \forall x{:}\tau.A \mid \ldots \\
\text{Terms} & M, N & ::= & \{c \leftarrow P \leftarrow \overline{d_i}\} \\
\ldots & & &
\end{array}
$$

Encoding

Definition

Validating our system by encoding functions in the process layer:

- Encoding of a term M is indexed by a result channel z – $[\![M]\!]_z$.
- Target calculus is a higher-order (i.e. process-passing) process calculus:

$$
\begin{array}{lllll}
\text{Types} & \tau, \sigma & ::= & \{\overline{d_i{:}B_i} \vdash c{:}A\} \\
\text{Sessions} & A, B & ::= & \forall x{:}\tau.A \mid \ldots \\
\text{Terms} & M, N & ::= & \{c \leftarrow P \leftarrow \overline{d_i}\} \\
\ldots & & &
\end{array}
$$

Functional:

$$[\![\Pi x{:}\tau.\sigma]\!] \triangleq \forall x{:}\{[\![\tau]\!]\}.[\![\sigma]\!]$$

Encoding

Definition

Validating our system by encoding functions in the process layer:

- Encoding of a term M is indexed by a result channel z – $[\![M]\!]_z$.
- Target calculus is a higher-order (i.e. process-passing) process calculus:

$$
\begin{array}{llll}
\text{Types} & \tau, \sigma & ::= & \{\overline{d_i{:}B_i} \vdash c{:}A\} \\
\text{Sessions} & A, B & ::= & \forall x{:}\tau.A \mid \ldots \\
\text{Terms} & M, N & ::= & \{c \leftarrow P \leftarrow \overline{d_i}\} \\
\ldots & & &
\end{array}
$$

Functional:

$$
[\![\Pi x{:}\tau.\sigma]\!] \triangleq \forall x{:}\{[\![\tau]\!]\}.[\![\sigma]\!] \qquad [\![\{\overline{d_i{:}B_i} \vdash c{:}A\}]\!] \triangleq \overline{[\![B_i]\!]} \multimap [\![A]\!]
$$

Encoding

Definition

Validating our system by encoding functions in the process layer:

- Encoding of a term M is indexed by a result channel z – $[\![M]\!]_z$.
- Target calculus is a higher-order (i.e. process-passing) process calculus:

$$
\begin{array}{llll}
\text{Types} & \tau, \sigma & ::= & \{\overline{d_i{:}B_i} \vdash c{:}A\} \\
\text{Sessions} & A, B & ::= & \forall x{:}\tau.A \mid \ldots \\
\text{Terms} & M, N & ::= & \{c \leftarrow P \leftarrow \overline{d_i}\} \\
\ldots & & &
\end{array}
$$

Functional:

$$
\begin{array}{lcl}
[\![\Pi x{:}\tau.\sigma]\!] & \triangleq & \forall x{:}\{[\![\tau]\!]\}.[\![\sigma]\!] \\
[\![\lambda x{:}\tau.\sigma]\!] & \triangleq & \lambda x{:}\{[\![\tau]\!]\}.[\![\sigma]\!]
\end{array}
\qquad
[\![\{\overline{d_i{:}B_i} \vdash c{:}A\}]\!] \triangleq \overline{[\![B_i]\!]} \multimap [\![A]\!]
$$

Encoding

Definition

Validating our system by encoding functions in the process layer:

- Encoding of a term M is indexed by a result channel z – $[\![M]\!]_z$.
- Target calculus is a higher-order (i.e. process-passing) process calculus:

$$
\begin{array}{llll}
\text{Types} & \tau, \sigma & ::= & \{\overline{d_i{:}B_i} \vdash c{:}A\} \\
\text{Sessions} & A, B & ::= & \forall x{:}\tau.A \mid \ldots \\
\text{Terms} & M, N & ::= & \{c \leftarrow P \leftarrow \overline{d_i}\} \\
\ldots & & &
\end{array}
$$

Functional:

$$
\begin{array}{lcllcl}
[\![\Pi x{:}\tau.\sigma]\!] & \triangleq & \forall x{:}\{[\![\tau]\!]\}.[\![\sigma]\!] & [\![\{\overline{d_i{:}B_i} \vdash c{:}A\}]\!] & \triangleq & \overline{[\![B_i]\!]} \multimap [\![A]\!] \\
[\![\lambda x{:}\tau.\sigma]\!] & \triangleq & \lambda x{:}\{[\![\tau]\!]\}.[\![\sigma]\!] & [\![\tau\ M]\!] & \triangleq & [\![\tau]\!]\,\{[\![M]\!]_c\}
\end{array}
$$

Encoding

Definition

Validating our system by encoding functions in the process layer:

- Encoding of a term M is indexed by a result channel z – $[\![M]\!]_z$.
- Target calculus is a higher-order (i.e. process-passing) process calculus:

$$
\begin{array}{llll}
\text{Types} & \tau, \sigma & ::= & \{\overline{d_i{:}B_i} \vdash c{:}A\} \\
\text{Sessions} & A, B & ::= & \forall x{:}\tau.A \mid \ldots \\
\text{Terms} & M, N & ::= & \{c \leftarrow P \leftarrow \overline{d_i}\} \\
\ldots & & &
\end{array}
$$

Functional:

$$
\begin{array}{lcllcl}
[\![\Pi x{:}\tau.\sigma]\!] & \triangleq & \forall x{:}\{[\![\tau]\!]\}.[\![\sigma]\!] & [\![\{\overline{d_i{:}B_i} \vdash c{:}A\}]\!] & \triangleq & \overline{[\![B_i]\!]} \multimap [\![A]\!] \\
[\![\lambda x{:}\tau.\sigma]\!] & \triangleq & \lambda x{:}\{[\![\tau]\!]\}.[\![\sigma]\!] & [\![\tau\ M]\!] & \triangleq & [\![\tau]\!]\,\{[\![M]\!]_c\}
\end{array}
$$

Session:

$$
[\![\forall x{:}\tau.A]\!] \triangleq \forall x{:}\{[\![\tau]\!]\}.[\![A]\!]
$$

Encoding

Definition

Validating our system by encoding functions in the process layer:

- Encoding of a term M is indexed by a result channel z – $[\![M]\!]_z$.
- Target calculus is a higher-order (i.e. process-passing) process calculus:

$$
\begin{array}{llll}
\text{Types} & \tau, \sigma & ::= & \{\overline{d_i{:}B_i} \vdash c{:}A\} \\
\text{Sessions} & A, B & ::= & \forall x{:}\tau.A \mid \ldots \\
\text{Terms} & M, N & ::= & \{c \leftarrow P \leftarrow \overline{d_i}\} \\
\ldots & & &
\end{array}
$$

Functional:

$$
\begin{array}{llll}
[\![\Pi x{:}\tau.\sigma]\!] & \triangleq & \forall x{:}\{[\![\tau]\!]\}.[\![\sigma]\!] & [\![\{\overline{d_i{:}B_i} \vdash c{:}A\}]\!] \triangleq \overline{[\![B_i]\!]} \multimap [\![A]\!] \\
[\![\lambda x{:}\tau.\sigma]\!] & \triangleq & \lambda x{:}\{[\![\tau]\!]\}.[\![\sigma]\!] & [\![\tau\ M]\!] \triangleq [\![\tau]\!]\,\{[\![M]\!]_c\}
\end{array}
$$

Session:

$$
[\![\forall x{:}\tau.A]\!] \triangleq \forall x{:}\{[\![\tau]\!]\}.[\![A]\!] \qquad [\![\exists x{:}\tau.A]\!] \triangleq \exists x{:}\{[\![\tau]\!]\}.[\![A]\!]
$$

Encoding

Definition – Terms

Terms:

$$[\![\lambda x{:}\tau.M]\!]_z \triangleq z(x{:}\{[\![\tau]\!]\}).[\![M]\!]_z$$

Encoding

Definition – Terms

Terms:

$$[\![\lambda x{:}\tau.M]\!]_z \triangleq z(x{:}\{[\![\tau]\!]\}).[\![M]\!]_z \qquad [\![M\,N]\!]_z \triangleq (\nu x)([\![M]\!]_x \mid x\langle\{[\![N]\!]_y\}\rangle.[x \leftrightarrow z])$$

Encoding

Definition – Terms

Terms:

$$[\![\lambda x{:}\tau.M]\!]_z \triangleq z(x{:}\{[\![\tau]\!]\}).[\![M]\!]_z \qquad [\![M\,N]\!]_z \triangleq (\nu x)([\![M]\!]_x \mid x\langle\{[\![N]\!]_y\}\rangle.[x \leftrightarrow z])$$

$$[\![x]\!]_z \triangleq y \leftarrow x; [y \leftrightarrow z]$$

Encoding

Definition – Terms

Terms:

$$\llbracket \lambda x{:}\tau.M \rrbracket_z \triangleq z(x{:}\{\llbracket \tau \rrbracket\}).\llbracket M \rrbracket_z \qquad \llbracket M\,N \rrbracket_z \triangleq (\nu x)(\llbracket M \rrbracket_x \mid x\langle\{\llbracket N \rrbracket_y\}\rangle.[x \leftrightarrow z])$$

$$\llbracket x \rrbracket_z \triangleq y \leftarrow x; [y \leftrightarrow z] \qquad \llbracket \{z \leftarrow P \leftarrow \overline{d_i}\} \rrbracket_z \triangleq z(d_0).\ldots.z(d_n).\llbracket P \rrbracket$$

- Encoding for processes is homomorphic, except composition of monadic terms.

Encoding

Definition – Terms

Terms:

$$\llbracket \lambda x{:}\tau.M \rrbracket_z \triangleq z(x{:}\{\llbracket \tau \rrbracket\}).\llbracket M \rrbracket_z \qquad \llbracket M\, N \rrbracket_z \triangleq (\nu x)(\llbracket M \rrbracket_x \mid x\langle\{\llbracket N \rrbracket_y\}\rangle.[x \leftrightarrow z])$$

$$\llbracket x \rrbracket_z \triangleq y \leftarrow x; [y \leftrightarrow z] \qquad \llbracket \{z \leftarrow P \leftarrow \overline{d_i}\} \rrbracket_z \triangleq z(d_0).\ldots.z(d_n).\llbracket P \rrbracket$$

- Encoding for processes is homomorphic, except composition of monadic terms.
- We can prove encoding is correct up-to definitional equality (no need to talk about behavioural equivalence).

Compositionality

- $\Psi; \Delta \vdash \llbracket M\{N/x\} \rrbracket_z = \llbracket M \rrbracket_z\{\{\llbracket N \rrbracket_y\}/x\} :: z{:}\llbracket A\{N/x\} \rrbracket$
- $\Psi; \Delta \vdash \llbracket P\{M/x\} \rrbracket :: z{:}\llbracket A\{M/x\} \rrbracket$ iff $\Psi; \Delta \vdash \llbracket P \rrbracket\{\{\llbracket M \rrbracket_c\}/x\} :: z{:}\llbracket A \rrbracket\{\{\llbracket M \rrbracket_c\}/x\}$

Encoding

Correctness

Preservation of Equality/Typing

- If $\Psi \vdash M = N : \tau$ then $\{[\![\Psi]\!]\}; \cdot \vdash [\![M]\!]_z = [\![N]\!]_z :: z{:}[\![\tau]\!]$
- If $\Psi \vdash M : \tau$ then $\{[\![\Psi]\!]\}; \cdot \vdash [\![M]\!]_z :: z{:}[\![\tau]\!]$

Encoding

Correctness

Preservation of Equality/Typing

- If $\Psi \vdash M = N : \tau$ then $\{[\![\Psi]\!]\}; \cdot \vdash [\![M]\!]_z = [\![N]\!]_z :: z{:}[\![\tau]\!]$
- If $\Psi \vdash M : \tau$ then $\{[\![\Psi]\!]\}; \cdot \vdash [\![M]\!]_z :: z{:}[\![\tau]\!]$

Operational Correspondence

If $\Psi \vdash M : \tau$ then:

(a) If $M \to M'$ then $[\![M]\!]_z \to^+ N$ with $\{[\![\Psi]\!]\}; \cdot; \cdot \vdash N = [\![M']\!]_z :: z{:}[\![\tau]\!]$ and

(b) If $[\![M]\!]_z \to P$ then $M \to N$ with $\{[\![\Psi]\!]\}; \cdot; \cdot \vdash [\![N]\!]_z = P :: z{:}[\![\tau]\!]$

Concluding Remarks

- We have introduced a language of dependent functions and processes.
- Dependencies "in both directions" + Type-level functions.
- Allows for both dependently-typed programming and reasoning.

Concluding Remarks

- We have introduced a language of dependent functions and processes.
- Dependencies "in both directions" + Type-level functions.
- Allows for both dependently-typed programming and reasoning.

Future Work:

- Implementation
- More on inductive/coinductive definitions.
- Higher-kinded reasoning, etc.