Curry-Howard Correspondences for Concurrency

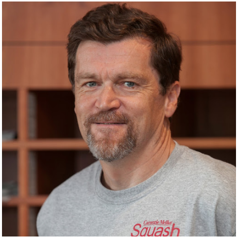
Overview and Recent Developments

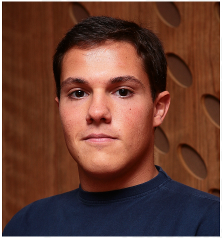
Jorge A. Pérez

university of groningen

University of Brasilia
July 21, 2015

Acknowledgments





This talk is based on works with/by Caires, Pfenning & Toninho:

- CONCUR'10 / Math. Str. in Comp. Science (In press)
- ESOP'12 / Information and Computation (In press)

This Talk

Using logic to reason about the correctness of software systems

This Talk

Using logic to reason about the correctness of communicating software systems

This Talk

Using linear logic to reason about the correctness of communicating software systems

Outline

Large-scale Software Infrastructures

- Massive collections of services – distributed software artifacts
  - ⋆ Heterogeneous, dynamic, extensible, composable, long-running
- Concurrent and communication-centered
  - ⋆ Services expose behavioral interfaces
  - ⋆ Complex interaction/coordination patterns among them
- Correctness is a combination of several issues, including:
  - ⋆ Protocol compatibility
  - ⋆ Resource usage
  - ⋆ Security and trustworthiness
- Building correct communicating software is difficult!
  - ⋆ A major societal challenge
  - ⋆ Costly, embarrassing errors still occur.

Behavioral Types

By classifying values, usual type systems are an effective basis for validating and verifying sequential programs

To reason about services, behavioral types classify interactions

- High-level representations of communication structures
- A compositional basis for (statically) checking service behavior
- Tied to programming abstractions which promote communication as a first-class concern

Behavioral Types

By classifying values, usual type systems are an effective basis for validating and verifying sequential programs

To reason about services, behavioral types classify interactions

- High-level representations of communication structures
- A compositional basis for (statically) checking service behavior
- Tied to programming abstractions which promote communication as a first-class concern

Behavioral Types

- Typically developed upon core programming models, such as process calculi
  - ⋆ Variants of the π-calculus [Milner, Parrow, & Walker, 89]
  - ⋆ Expressive core programming models; adequate for investigation
- Formal specification languages, based on communication
  - ⋆ Centered around interactions of partners with reciprocal roles
  - ⋆ Strong ties with established theories (automata, logic, types)
  - ⋆ Clear linkage with validation methods
  - ⋆ Precise notions of runtime correctness

Session Types (1)

Seminal type-based approach to the analysis of structured communications [Honda, Vasconcelos, Kubo (1998)]

- Communication protocols structured into sessions
- Concurrent processes communicating through session channels
- Disciplined interactive behavior, abstracted as session types

Session Types (2)

Session specifications are usually given as π-calculus processes

- Actions always occur in dual pairs
- New sessions created by invoking shared servers
- Concurrency in the simultaneous execution of sessions
- Mobility in the exchange of session and server names

Correctness Guarantees for Specifications

- Adhere to their ascribed session protocols - Fidelity
- Do not feature runtime errors – Safety
- Do not get stuck – Progress / Lock-Freedom
- Do not have infinite reduction sequences – Termination

Example: An E-commerce Service

The Service: Informal Description

1. Receive an item description from a client
2. Return a boolean confirming availability
3. Offer a choice: save the transaction (and pay later) OR conclude the transaction and proceed with payment.

The Service As a Session Type

$$\mathsf{Store} \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\mathtt{later} : \mathsf{SaveStore} \mathbin{\&} \mathtt{now} : \mathsf{PayStore})$$

The Client As a Session Type (Dual to Store)

$$\mathsf{Client} \triangleq \mathrm{item} \otimes \mathrm{bool} \multimap (\mathtt{later} : \mathsf{SaveCli} \oplus \mathtt{now} : \mathsf{PayCli})$$

Example: An E-commerce Service

The Service: Informal Description

1. Receive an item description from a client
2. Return a boolean confirming availability
3. Offer a choice: save the transaction (and pay later) OR conclude the transaction and proceed with payment.

The Service As a Session Type

$$\mathsf{Store} \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\mathtt{later} : \mathsf{SaveStore} \,\&\, \mathtt{now} : \mathsf{PayStore})$$

The Client As a Session Type (Dual to Store)

$$\mathsf{Client} \triangleq \mathrm{item} \otimes \mathrm{bool} \multimap (\mathtt{later} : \mathsf{SaveCli} \oplus \mathtt{now} : \mathsf{PayCli})$$

Example: An E-commerce Service

The Service: Informal Description

1. Receive an item description from a client
2. Return a boolean confirming availability
3. Offer a choice: save the transaction (and pay later) OR conclude the transaction and proceed with payment.

The Service As a Session Type

$$\mathsf{Store} \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\mathtt{later} : \mathsf{SaveStore} \;\&\; \mathtt{now} : \mathsf{PayStore})$$

The Client As a Session Type (Dual to Store)

$$\mathsf{Client} \triangleq \mathrm{item} \otimes \mathrm{bool} \multimap (\mathtt{later} : \mathsf{SaveCli} \oplus \mathtt{now} : \mathsf{PayCli})$$

Outline

Logical Foundations for Session Types

A Concurrent Interpretation of Linear Logic [Caires & Pfenning, 2010]

Based on dual intuitionistic linear logic (DILL) [cf. Barber&Plotkin]

propositions	$\leftrightarrow$	session types
sequent proofs	$\leftrightarrow$	π-calculus processes
cut elimination	$\leftrightarrow$	process communication

Main Features

- Clear account of resource usage policies in concurrency
- Session fidelity, runtime safety, global progress “for free”
- Excellent basis for generalizations and extensions

Outline

A Synchronous π-calculus

$P, Q ::=$	$\overline{x}\, z.P$	send z on x, proceed as P
$\mid$	$x(y).P$	receive z on x, proceed as $P\{{}^{z}\!/\!{}_{y}\}$
$\mid$	$!x(y).P$	replicated server at x
$\mid$	$x.\mathtt{case}(P, Q)$	branching: offers a choice at x
$\mid$	$x.\mathtt{inl}; P$	select left at x, continue as P
$\mid$	$x.\mathtt{inr}; P$	select right at x, continue as P
$\mid$	$[x \leftrightarrow y]$	forwarder, equates names x and y
$\mid$	$P \mid Q$	parallel composition
$\mid$	$(\boldsymbol{\nu} y)P$	name restriction
$\mid$	$\mathbf{0}$	inaction

Notation: We write $\overline{x}(y)$ to stand for the bound output $(\boldsymbol{\nu} y)\overline{x}\, y$.

A Synchronous π-calculus

P, Q	$::=$	$\overline{x}\, z.P$	send z on x, proceed as P
	$\mid$	$x(y).P$	receive z on x, proceed as $P\{z/y\}$
	$\mid$	$!x(y).P$	replicated server at x
	$\mid$	$x \triangleright \{\mathtt{l}_1{:}P_1, \ldots, \mathtt{l}_n{:}P_n\}$	branching: offers a choice at x
	$\mid$	$x \triangleleft \mathtt{l}_j; P$	select label $\mathtt{l}_j$ at x, continue as P
	$\mid$	$[x \leftrightarrow y]$	forwarder, equates names x and y
	$\mid$	$P \mid Q$	parallel composition
	$\mid$	$(\boldsymbol{\nu} y)P$	name restriction
	$\mid$	$\mathbf{0}$	inaction

Notation: We write $\overline{x}(y)$ to stand for the bound output $(\boldsymbol{\nu} y)\overline{x}\, y$.

Operational Semantics

- Reduction gives the behavior of a process on its own:

$$
\begin{array}{rcl}
\overline{x}\,y.Q \mid x(z).P & \longrightarrow & Q \mid P\{y/z\} \\
\overline{x}\,y.Q \mid !x(z).P & \longrightarrow & Q \mid P\{y/z\} \mid !x(z).P \\
x.\mathtt{inr}; P \mid x.\mathsf{case}(Q, R) & \longrightarrow & P \mid R \\
x.\mathtt{inl}; P \mid x.\mathsf{case}(Q, R) & \longrightarrow & P \mid Q \\
(\boldsymbol{\nu} x)([x \leftrightarrow y] \mid P) & \longrightarrow & P\{y/x\} \\
Q \longrightarrow Q' & \Rightarrow & P \mid Q \longrightarrow P \mid Q' \\
P \longrightarrow Q & \Rightarrow & (\boldsymbol{\nu} y)P \longrightarrow (\boldsymbol{\nu} y)Q
\end{array}
$$

Closed under structural congruence, noted $\equiv$.

- A standard LTS with labels for selection/choice constructs:

$$
\lambda \quad ::= \quad \tau \;\;\Big|\;\; x(y) \;\;\Big|\;\; x \triangleleft \mathtt{l} \;\;\Big|\;\; \overline{x}\,y \;\;\Big|\;\; \overline{x}(y) \;\;\Big|\;\; \overline{x \triangleleft \mathtt{l}}
$$

Strong transitions $\xrightarrow{\lambda}$ and weak transitions $\xRightarrow{\lambda}$, as usual.

Session Types as Linear Logic Propositions

The syntax of types coincides with dual intuitionistic linear logic.
Propositions/types (A, B, C, T) are assigned to names:

$x : A \otimes B$	Output an A along x, behave as B on x
$x : A \multimap B$	Input an A along x, behave as B on x
$x : !A$	Persistently offer A along x
$x : A \mathbin{\&} B$	Offer both A and B along x
$x : A \oplus B$	Select either A or B along x
$x : \mathbf{1}$	Terminated interaction on x

Session Types as Linear Logic Propositions

The syntax of types coincides with dual intuitionistic linear logic.
Propositions/types (A, B, C, T) are assigned to names:

$x : A \otimes B$	Output an A along x, behave as B on x
$x : A \multimap B$	Input an A along x, behave as B on x
$x : !A$	Persistently offer A along x
$x : \&\{\mathtt{l}_1{:}A_1, \ldots, \mathtt{l}_n{:}A_n\}$	Offer $A_1, \ldots, A_n$ along x
$x : \oplus\{\mathtt{l}_1{:}A_1, \ldots, \mathtt{l}_n{:}A_n\}$	Select one of $A_1, \ldots, A_n$ along x
$x : \mathbf{1}$	Terminated interaction on x

Type Judgments: Intuitions

$$P :: z : C$$

Process P offers *behavior* C at *name* z

when composed with

processes offering A_1 at $x_1, \ldots, A_n$ at x_n

Examples

$\Delta \vdash$	$P :: z : \mathbf{1}$	P offers nothing relying on behaviors Δ
$\cdot \vdash$	$Q :: z : !A$	Q is an autonomous replicated server
$x : A \otimes B \vdash$	$R :: z : C$	R requires A, B on x to offer $z : C$

Type Judgments: Intuitions

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: z : C$$

Process P offers *behavior* C at *name* z
when composed with
processes offering A_1 at x_1, …, A_n at x_n

Examples

$\Delta \vdash$	$P :: z : \mathbf{1}$	P offers nothing relying on behaviors Δ
$\cdot \vdash$	$Q :: z : !A$	Q is an autonomous replicated server
$x : A \otimes B \vdash$	$R :: z : C$	R requires A, B on x to offer $z : C$

Type Judgments: Intuitions

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: z : C$$

Process P offers *behavior* C at *name* z
when composed with
processes offering A_1 at x_1, …, A_n at x_n

Examples

$\Delta \vdash$	$P :: z : \mathbf{1}$	P offers nothing relying on behaviors Δ
$\cdot \vdash$	$Q :: z : !A$	Q is an autonomous replicated server
$x : A \otimes B \vdash$	$R :: z : C$	R requires A, B on x to offer $z : C$

Type Judgments, Actually

Dependencies as two collections of type assignments, Γ and Δ:

$$\underbrace{u_1 : A_1, \ldots, u_n : A_n}_{\Gamma} \; ; \; \underbrace{x_1 : B_1, \ldots, x_k : B_k}_{\Delta} \vdash P :: z : C$$

- Γ specifies shared services A_i along u_i
- Δ specifies linear services B_j along x_j [no weakening or contraction]

(u_i, x_j, z pairwise distinct.)

Example: PDF Conversion Service

Receive a file and then either return a PDF version of it OR quit:

$$\text{Converter} \triangleq \mathrm{file} \multimap \big((\mathrm{PDF} \otimes \mathbf{1}) \,\&\, \mathbf{1}\big)$$

- A process which offers a linear conversion service:

$$Server \triangleq x(f).x \triangleright \{\mathtt{conv} : \overline{x}(y).C_{(f,y)}\,,\ \mathtt{quit} : Q\}$$

- A user which depends on the server:

$$User \triangleq \overline{x}(txt).x \triangleleft \mathtt{conv}; x(pdf).R$$

- Next, we will see how server and user can be composed:

$$\frac{\cdot \vdash Server :: x : \text{Converter} \qquad x : \text{Converter} \vdash User :: z : A}{\cdot \vdash (\boldsymbol{\nu} x)(Server \mid User) :: z : A}$$

Outline

Typing Rules

The logic correspondence induces right and left typing rules:

- Right rules detail how a process can implement the behavior described by the given connective
- Left rules explain how a process may use a session of a given type

Cut rules in sequent calculus are interpreted as well-typed process composition, based on both restriction and parallel composition.

Some Typing Rules

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

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta' \vdash Q :: x : B}{\Gamma; \Delta, \Delta' \vdash \overline{x}(y).(P \mid Q) :: x : A \otimes B}$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash P :: T}{\Gamma; \Delta, x : A \otimes B \vdash x(y).P :: T}$$

$$\frac{\Gamma; \Delta \vdash P :: x : A \qquad \Gamma; \Delta \vdash Q :: x : B}{\Gamma; \Delta \vdash x.\mathsf{case}(P, Q) :: x : A \,\&\, B}$$

$$\frac{\Gamma; \Delta, x : A \vdash P :: T}{\Gamma; \Delta, x : A \,\&\, B \vdash x.\mathtt{inl}; P :: T}$$

Some Typing Rules

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

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta' \vdash Q :: x : B}{\Gamma; \Delta, \Delta' \vdash \overline{x}(y).(P \mid Q) :: x : A \otimes B}$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash P :: T}{\Gamma; \Delta, x : A \otimes B \vdash x(y).P :: T}$$

$$\frac{\Gamma; \Delta \vdash P :: x : A \qquad \Gamma; \Delta \vdash Q :: x : B}{\Gamma; \Delta \vdash x.\mathsf{case}(P, Q) :: x : A \,\&\, B}$$

$$\frac{\Gamma; \Delta, x : A \vdash P :: T}{\Gamma; \Delta, x : A \,\&\, B \vdash x.\mathtt{inl}; P :: T}$$

Some Typing Rules

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

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta' \vdash Q :: x : B}{\Gamma; \Delta, \Delta' \vdash \overline{x}(y).(P \mid Q) :: x : A \otimes B}$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash P :: T}{\Gamma; \Delta, x : A \otimes B \vdash x(y).P :: T}$$

$$\frac{\Gamma; \Delta \vdash P :: x : A \qquad \Gamma; \Delta \vdash Q :: x : B}{\Gamma; \Delta \vdash x.\mathsf{case}(P, Q) :: x : A \,\&\, B}$$

$$\frac{\Gamma; \Delta, x : A \vdash P :: T}{\Gamma; \Delta, x : A \,\&\, B \vdash x.\mathtt{inl}; P :: T}$$

Some Typing Rules

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

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta' \vdash Q :: x : B}{\Gamma; \Delta, \Delta' \vdash \overline{x}(y).(P \mid Q) :: x : A \otimes B}$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash P :: T}{\Gamma; \Delta, x : A \otimes B \vdash x(y).P :: T}$$

$$\frac{\Gamma; \Delta \vdash P :: x : A \qquad \Gamma; \Delta \vdash Q :: x : B}{\Gamma; \Delta \vdash x.\mathsf{case}(P, Q) :: x : A \,\&\, B}$$

$$\frac{\Gamma; \Delta, x : A \vdash P :: T}{\Gamma; \Delta, x : A \,\&\, B \vdash x.\mathtt{inl}; P :: T}$$

Some Typing Rules

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

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta' \vdash Q :: x : B}{\Gamma; \Delta, \Delta' \vdash \overline{x}(y).(P \mid Q) :: x : A \otimes B}$$

$$\frac{\Gamma; \Delta, y : A, x : B \vdash P :: T}{\Gamma; \Delta, x : A \otimes B \vdash x(y).P :: T}$$

$$\frac{\Gamma; \Delta \vdash P :: x : A \qquad \Gamma; \Delta \vdash Q :: x : B}{\Gamma; \Delta \vdash x.\mathsf{case}(P, Q) :: x : A \,\&\, B}$$

$$\frac{\Gamma; \Delta, x : A \vdash P :: T}{\Gamma; \Delta, x : A \,\&\, B \vdash x.\mathtt{inl}; P :: T}$$

Typing Composition

Linear Composition

Cut as composition principle for linear services:

$$\frac{\Gamma; \Delta \vdash P :: x : A \qquad \Gamma; \Delta', x : A \vdash Q :: T}{\Gamma; \Delta, \Delta' \vdash (\boldsymbol{\nu} x)(P \mid Q) :: T}$$

Shared Composition

Cut! as composition principle for shared services:

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

Cut as Process Reduction: Linear Case

$$\dfrac{\dfrac{\Delta_1 \vdash P_1 :: y{:}A \qquad \Delta_2 \vdash P_2 :: x{:}B}{\Delta_1, \Delta_2 \vdash \overline{x}(y).(P_1 \mid P_2) :: x{:}A \otimes B} \quad \dfrac{\Delta_3, y{:}A, x{:}B \vdash Q :: T}{\Delta_3, x{:}A \otimes B \vdash x(y).Q :: T}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)(\overline{x}(y).(P_1 \mid P_2) \mid x(y).Q) :: T}$$

$$\longrightarrow$$

$$\dfrac{\Delta_2 \vdash P_2 :: x{:}B \quad \dfrac{\Delta_1 \vdash P_1 :: y{:}A \qquad \Delta_3, y{:}A, x{:}B \vdash Q :: T}{\Delta_1, \Delta_3, x{:}B \vdash (\boldsymbol{\nu} y)(P_1 \mid Q) :: T}}{\Delta_1, \Delta_2, \Delta_3 \vdash (\boldsymbol{\nu} x)(P_2 \mid (\boldsymbol{\nu} y)(P_1 \mid Q)) :: T}$$

Cut as Process Reduction: Shared Case

$$\dfrac{\Gamma; \cdot \vdash P :: x{:}A \qquad \dfrac{\Gamma, u{:}A; \Delta, x{:}A \vdash Q :: T}{\Gamma, u{:}A; \Delta \vdash \overline{u}(x).Q :: T}\,\text{copy}}{\Gamma; \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid \overline{u}(x).Q) :: T}\,\text{cut}^{!}$$

$$\longrightarrow$$

$$\dfrac{\Gamma; \cdot \vdash P :: x{:}A \qquad \dfrac{\Gamma; \cdot \vdash P :: x{:}A \qquad \Gamma, u{:}A; \Delta, x{:}A \vdash Q :: T}{\Gamma; \Delta, x{:}A \vdash (\boldsymbol{\nu} u)(!u(x).P \mid Q) :: T}\,\text{cut}^{!}}{\Gamma; \Delta \vdash (\boldsymbol{\nu} x)(P \mid (\boldsymbol{\nu} u)(!u(x).P \mid Q)) :: T}\,\text{cut}$$

Properties of the Type System

Theorem (Type Preservation)

If $\Gamma; \Delta \vdash P :: z : A$ *and* $P \longrightarrow Q$ *then* $\Gamma; \Delta \vdash Q :: z : A$.

- Process reductions map to principal cut reductions
- Derived properties: communication safety and session fidelity.

For any P, define $live(P)$ iff $P \equiv (\boldsymbol{\nu}\overline{n})(\pi.Q \mid R)$ for some $\pi.Q, R, \overline{n}$ where $\pi.Q$ is a non-replicated guarded process.

Theorem (Global Progress / Deadlock Avoidance)

If $\cdot; \cdot \vdash P :: z : \mathbf{1}$ *and* $live(P)$ *then exists a* Q *such that* $P \longrightarrow Q$.

Properties of the Type System

Theorem (Type Preservation)

If $\Gamma; \Delta \vdash P :: z : A$ *and* $P \longrightarrow Q$ *then* $\Gamma; \Delta \vdash Q :: z : A$.

- Process reductions map to principal cut reductions
- Derived properties: communication safety and session fidelity.

For any P, define $live(P)$ iff $P \equiv (\boldsymbol{\nu}\overline{n})(\pi.Q \mid R)$ for some $\pi.Q, R, \overline{n}$ where $\pi.Q$ is a non-replicated guarded process.

Theorem (Global Progress / Deadlock Avoidance)

If $\cdot; \cdot \vdash P :: z : \mathbf{1}$ *and* $live(P)$ *then exists a* Q *such that* $P \longrightarrow Q$.

Outline

Outline

Linear LRs for Session Types: Highlights

- Logical relations (LRs): well-establshed method in the functional setting [cf. the simply-typed λ-calculus]
- We instantiate the method with our *linear* session type structure, to establish termination and confluence of well-typed processes.
- Practical significance: enhanced session predictability.

Linear LRs for Session Types: Definitions

Termination and Confluence

- P terminates, noted $P\Downarrow$, if either $P \not\longrightarrow$ or for any P' such that $P \longrightarrow P'$ we have that $P' \Longrightarrow P'' \not\longrightarrow$.
- P is confluent if for any P_1, P_2 such that $P \Longrightarrow P_1$ and $P \Longrightarrow P_2$, there exists a P' such that $P_1 \Longrightarrow P'$ and $P_2 \Longrightarrow P'$.

The Logical Predicate

- A sequent-indexed family of sets of processes.
  For each $\Gamma; \Delta \vdash T$, a set of processes $\mathcal{L}[\Gamma; \Delta \vdash T\,]$
- Defined inductively: the base case is $\mathcal{L}[\cdot; \cdot \vdash T\,]$, written $\mathcal{L}[T\,]$
  The inductive case $(\Gamma, \Delta \neq \emptyset)$ uses typed process composition.

Linear LRs for Session Types: Definitions

Termination and Confluence

- P terminates, noted $P\Downarrow$, if either $P \not\longrightarrow$ or for any P' such that $P \longrightarrow P'$ we have that $P' \Longrightarrow P'' \not\longrightarrow$.
- P is confluent if for any P_1, P_2 such that $P \Longrightarrow P_1$ and $P \Longrightarrow P_2$, there exists a P' such that $P_1 \Longrightarrow P'$ and $P_2 \Longrightarrow P'$.

The Logical Predicate

- A sequent-indexed family of sets of processes.
  For each $\Gamma; \Delta \vdash T$, a set of processes $\mathcal{L}[\Gamma; \Delta \vdash T\,]$
- Defined inductively: the base case is $\mathcal{L}[\cdot; \cdot \vdash T\,]$, written $\mathcal{L}[T\,]$
  The inductive case ($\Gamma, \Delta \neq \emptyset$) uses typed process composition.

The Logical Predicate

Inductive Case (Excerpt)

$P \in \mathcal{L}[\Gamma; \Delta, y{:}A \vdash T]$ iff $\forall R \in \mathcal{L}[y : A].(\boldsymbol{\nu} y)(R \mid P) \in \mathcal{L}[\Gamma; \Delta \vdash T]$

Base Case (Excerpt)

$\mathcal{L}[T]$ is the set of all P such that $P\Downarrow$ and $\cdot; \cdot \vdash P :: T$ and

$$P \in \mathcal{L}[z : \mathbf{1}] \text{ iff } \forall P'.(P \Longrightarrow P' \wedge P' \not\longrightarrow) \text{ implies } P' \equiv_! \mathbf{0}$$

$$P \in \mathcal{L}[z : A \multimap B] \text{ iff } \forall P' y.(P \overset{z(y)}{\Longrightarrow} P') \text{ implies } \forall Q \in \mathcal{L}[y : A].(\boldsymbol{\nu} y)(P' \mid Q) \in \mathcal{L}[z : B]$$

$$P \in \mathcal{L}[z : A \otimes B] \text{ iff } \forall P' y.(P \overset{\overline{z}(y)}{\Longrightarrow} P') \text{ implies } \exists P_1, P_2.(P' \equiv_! P_1 \mid P_2 \;\wedge\; P_1 \in \mathcal{L}[y : A] \;\wedge\; P_2 \in \mathcal{L}[z : B])$$

Proving Termination

Lemma (Fundamental Lemma)

Let P be a process. If $\Gamma; \Delta \vdash P :: T$ then $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$.

[Proof by induction on typing, using a few closure properties for $\mathcal{L}[T]$.]

As a direct consequence of this lemma, we have:

Theorem (Well-typed Processes Terminate)

If $\Gamma; \Delta \vdash P :: T$ then $P\Downarrow$.

(The proof of confluence follows very similar lines.)

Outline

Context Bisimilarity ($\approx$): Intuitions

- A behavioral equivalence for session-typed processes.
- Given two processes P and Q, typed under the same environments, we write

$$\Gamma; \Delta \vdash P \approx Q :: z : C$$

- Intuitively, P and Q behave the same at $\Gamma; \Delta \vdash z : C$.
- Formally: there is a type-respecting relation $\mathcal{R}$ which contains (P, Q) *and* which is a context bisimulation.

Context Bisimulation: Key Ideas

- Context bisimulation is defined inductively on Γ, Δ, C:
  - ⋆ Generalizes the predicate for LRs
  - ⋆ The base case follows the nature of C
  - ⋆ The inductive case uses typed composition (linear and shared)
- A weak bisimulation: action $\xrightarrow{\lambda}$ is matched by $\overset{\lambda}{\Longrightarrow}$
  But termination ensures reductions in weak actions are finite!

Context Bisimulation: Key Ideas

A symmetric, type-respecting relation $\mathcal{R}$ is a context bisimulation if

Inductive case (excerpt)

If $\boxed{\Gamma; \Delta, y{:}A \vdash P \mathrel{\mathcal{R}} Q :: T}$ then, $\forall R.\ \vdash R :: y{:}A$,

$$\boxed{\Gamma; \Delta \vdash (\boldsymbol{\nu} y)(R \mid P) \mathrel{\mathcal{R}} (\boldsymbol{\nu} y)(R \mid Q) :: T}.$$

Base case (excerpt)

- $\boxed{\vdash P \mathrel{\mathcal{R}} Q :: x : A \multimap B}$ implies that $\forall P'.\ P \xrightarrow{x(y)} P'$, $\exists Q'.\ Q \overset{x(y)}{\Longrightarrow} Q'$ and $\forall R.\ \vdash R :: y : A$, $\boxed{\vdash (\boldsymbol{\nu} y)(P' \mid R) \mathrel{\mathcal{R}} (\boldsymbol{\nu} y)(Q' \mid R) :: x : B}$
- $\boxed{\vdash P \mathrel{\mathcal{R}} Q :: x : !A}$ implies that $\forall P'.\ P \xrightarrow{x(z)} P'$, $\exists Q'.\ Q \overset{x(z)}{\Longrightarrow} Q'$ and $\forall R.\ \cdot ; y : A \vdash R :: - : \mathbf{1}$ $\boxed{\vdash (\boldsymbol{\nu} y)(P' \mid R) \mathrel{\mathcal{R}} (\boldsymbol{\nu} y)(Q' \mid R) :: x : !A}$

Context bisimilarity ($\approx$) is the union of all context bisimulations.

Context Bisimulation: Key Ideas

A symmetric, type-respecting relation $\mathcal{R}$ is a context bisimulation if

Inductive case (excerpt)

If $\boxed{\Gamma; \Delta, y{:}A \vdash P \;\mathcal{R}\; Q :: T}$ then, $\forall R.\ \vdash R :: y{:}A$,

$$\boxed{\Gamma; \Delta \vdash (\boldsymbol{\nu} y)(R \mid P) \;\mathcal{R}\; (\boldsymbol{\nu} y)(R \mid Q) :: T}.$$

Base case (excerpt)

- $\boxed{\vdash P \;\mathcal{R}\; Q :: x : A \multimap B}$ implies that $\forall P'.\ P \xrightarrow{x(y)} P'$, $\exists Q'.\ Q \overset{x(y)}{\Longrightarrow} Q'$ and $\forall R.\ \vdash R :: y : A$, $\boxed{\vdash (\boldsymbol{\nu} y)(P' \mid R) \;\mathcal{R}\; (\boldsymbol{\nu} y)(Q' \mid R) :: x : B}$
- $\boxed{\vdash P \;\mathcal{R}\; Q :: x :!A}$ implies that $\forall P'.\ P \xrightarrow{x(z)} P'$, $\exists Q'.\ Q \overset{x(z)}{\Longrightarrow} Q'$ and $\forall R.\ \cdot ; y : A \vdash R :: - : \mathbf{1}$ $\boxed{\vdash (\boldsymbol{\nu} y)(P' \mid R) \;\mathcal{R}\; (\boldsymbol{\nu} y)(Q' \mid R) :: x :!A}$

Context bisimilarity ($\approx$) is the union of all context bisimulations.

Context Bisimulation: Key Ideas

A symmetric, type-respecting relation $\mathcal{R}$ is a context bisimulation if

Inductive case (excerpt)

If $\boxed{\Gamma; \Delta, y{:}A \vdash P \; \mathcal{R} \; Q :: T}$ then, $\forall R.\ \vdash R :: y{:}A$,

$$\boxed{\Gamma; \Delta \vdash (\boldsymbol{\nu} y)(R \mid P) \; \mathcal{R} \; (\boldsymbol{\nu} y)(R \mid Q) :: T}.$$

Base case (excerpt)

- $\boxed{\vdash P \; \mathcal{R} \; Q :: x : A \multimap B}$ implies that $\forall P'.\ P \xrightarrow{x(y)} P'$,
  $\exists Q'.\ Q \stackrel{x(y)}{\Longrightarrow} Q'$ and $\forall R.\ \vdash R :: y : A$,
  $\boxed{\vdash (\boldsymbol{\nu} y)(P' \mid R) \, \mathcal{R} \, (\boldsymbol{\nu} y)(Q' \mid R) :: x : B}$
- $\boxed{\vdash P \; \mathcal{R} \; Q :: x :!A}$ implies that $\forall P'.\ P \xrightarrow{x(z)} P'$,
  $\exists Q'.\ Q \stackrel{x(z)}{\Longrightarrow} Q'$ and $\forall R.\ \ \cdot\, ; y : A \vdash R :: - : \mathbf{1}$
  $\boxed{\vdash (\boldsymbol{\nu} y)(P' \mid R) \; \mathcal{R} \; (\boldsymbol{\nu} y)(Q' \mid R) :: x :!A}$

Context bisimilarity ($\approx$) is the union of all context bisimulations.

Context Bisimilarity: Properties

Context bisimilarity enjoys the following properties:

- Is an equivalence
- Is a contextual relation, i.e., a congruence wrt typed contexts.
- Enjoys τ-inertness:
  If $\Gamma; \Delta \vdash P :: T$ and $P \longrightarrow P'$ then $\Gamma; \Delta \vdash P \approx P' :: T$.

Application: Session Type Isomorphisms

Types A, B are isomorphic if there are proofs of $B \vdash A$ and $A \vdash B$ which compose to the identity.

In our case:

- Useful as transformations of service interfaces
- Validation of basic logic principles. E.g. $A \otimes B \simeq B \otimes A$
- Natural definition in our setting, via context bisimilarity

Application: Session Type Isomorphisms

Types A, B are isomorphic if there are proofs of $B \vdash A$ and $A \vdash B$ which compose to the identity.

In our case:

- Useful as transformations of service interfaces
- Validation of basic logic principles. E.g. $A \otimes B \simeq B \otimes A$
- Natural definition in our setting, via context bisimilarity

Session Type Isomorphisms

We write $P^{\langle x,y \rangle}$ for a process P with free names x, y.

Definition

Session types A, B are called isomorphic, noted $A \simeq B$,
if for any x, y, z there exist processes $P^{\langle x,y \rangle}$ and $Q^{\langle y,x \rangle}$ such that:

1. $\cdot\,; x : A \vdash P^{\langle x,y \rangle} :: y : B$
2. $\cdot\,; y : B \vdash Q^{\langle y,x \rangle} :: x : A$
3. $\cdot\,; x : A \vdash (\boldsymbol{\nu} y)(P^{\langle x,y \rangle} \mid Q^{\langle y,z \rangle}) \approx [x \leftrightarrow z] :: z : A$
4. $\cdot\,; y : B \vdash (\boldsymbol{\nu} x)(Q^{\langle y,x \rangle} \mid P^{\langle x,z \rangle}) \approx [y \leftrightarrow z] :: z : B$

Type Isomorphisms: Symmetry of $\otimes$

Theorem

Let A, B be any session type. Then $A \otimes B \simeq B \otimes A$.

This does not mean that $P :: x : A \otimes B$ implies $P :: x : B \otimes A$!
It only implies that a suitable "coercion" exists:

$$\dfrac{\dfrac{\dfrac{}{x : B \vdash [x \leftrightarrow n] :: n : B}\ (\mathsf{Tid}) \qquad \dfrac{}{u : A \vdash [u \leftrightarrow y] :: y : A}\ (\mathsf{Tid})}{u : A, x : B \vdash \overline{y}(n).([x \leftrightarrow n] \mid [u \leftrightarrow y]) :: y : B \otimes A}\ (\mathsf{T}\otimes\mathsf{R})}{x : A \otimes B \vdash x(u).\overline{y}(n)([x \leftrightarrow n] \mid [u \leftrightarrow y]) :: y : B \otimes A}\ (\mathsf{T}\otimes\mathsf{L})$$

Note:

- Proofs combine type preservation, progress, termination.
- Other isomorphisms are handled analogously.

Type Isomorphisms: Symmetry of $\otimes$

Theorem

Let A, B be any session type. Then $A \otimes B \simeq B \otimes A$.

This does not mean that $P :: x : A \otimes B$ implies $P :: x : B \otimes A$!
It only implies that a suitable "coercion" exists:

$$\dfrac{\dfrac{}{x : B \vdash [x \leftrightarrow n] :: n : B}\,(\mathsf{Tid}) \quad \dfrac{}{u : A \vdash [u \leftrightarrow y] :: y : A}\,(\mathsf{Tid})}{\dfrac{u : A, x : B \vdash \overline{y}(n).([x \leftrightarrow n] \mid [u \leftrightarrow y]) :: y : B \otimes A}{x : A \otimes B \vdash x(u).\overline{y}(n)([x \leftrightarrow n] \mid [u \leftrightarrow y]) :: y : B \otimes A}\,(\mathsf{T}{\otimes}\mathsf{L})}\,(\mathsf{T}{\otimes}\mathsf{R})$$

Note:

- Proofs combine type preservation, progress, termination.
- Other isomorphisms are handled analogously.

Type Isomorphisms: Symmetry of $\otimes$

Theorem

Let A, B be any session type. Then $A \otimes B \simeq B \otimes A$.

This does not mean that $P :: x : A \otimes B$ implies $P :: x : B \otimes A$!
It only implies that a suitable "coercion" exists:

$$\dfrac{\dfrac{}{x : B \vdash [x \leftrightarrow n] :: n : B}\ (\mathsf{Tid}) \qquad \dfrac{}{u : A \vdash [u \leftrightarrow y] :: y : A}\ (\mathsf{Tid})}{\dfrac{u : A, x : B \vdash \overline{y}(n).([x \leftrightarrow n] \mid [u \leftrightarrow y]) :: y : B \otimes A}{x : A \otimes B \vdash x(u).\overline{y}(n)([x \leftrightarrow n] \mid [u \leftrightarrow y]) :: y : B \otimes A}\ (\mathsf{T}{\otimes}\mathsf{L})}\ (\mathsf{T}{\otimes}\mathsf{R})$$

Note:

- Proofs combine type preservation, progress, termination.
- Other isomorphisms are handled analogously.

Outline

Outline

Communications are Domain-Aware

- Services are nowadays offered virtualized in third-party platforms
  Communications must routinely span diverse domains
  (e.g. software and hardware domains, virtual organizations)
- Domains may influence structured interactive behavior
  L) Actions depend on the domains to which partners belong
  (e.g. domain-based capabilities/resources)
  G) Connectedness among domains enables communications
  (e.g., domain-based access control)
- Partners have local/partial visions of domain architectures
  (useful to enforce modularity, platform independence, security)
- The status of domains in structured communications unexplored

The Need for Domain-Awareness

Our Example, Revisited

A store receives an item that a client adds to her shopping cart. The store confirms availability, and then offers a choice:

$$\mathsf{Store} \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\texttt{later} : \mathsf{SaveStore} \mathbin{\&} \texttt{now} : \mathsf{PayStore})$$

Domain-related issues

- A client's sensitive data should be requested only after both partners move to a trusted domain (e.g. an `https` connection)
- Dually, the e-commerce platform should not allow client accesses to its payment domain in insecure ways

The Need for Domain-Awareness

Our Example, Revisited

A store receives an item that a client adds to her shopping cart. The store confirms availability, and then offers a choice:

$$\mathsf{Store} \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\mathtt{later} : \mathsf{SaveStore} \,\&\, \mathtt{now} : \mathsf{PayStore})$$

Domain-related issues are **hard to express**:

- A client's sensitive data should be requested only after both partners move to a trusted domain (e.g. an `https` connection)
- Dually, the e-commerce platform should not allow client accesses to its payment domain in insecure ways

Our Proposal: Domain-Aware Sessions

How to enhance session interfaces with domain-related information?

- Interplay between communication and domain-awareness
- Domains useful in both process specifications and type structure
- Enforcing correctness (preservation, progress, termination)

A concurrent interpretation of LL with **hybrid connectives**

- Modal worlds $w, w_1, \ldots$ as domains for distributed processes
- At the type level, hybrid connective $@_w$ as session migration
- At the process level, prefixes for domain-tagged channel passing
- Parametric accessibility relation governs movement

Our Proposal: Domain-Aware Sessions

How to enhance session interfaces with domain-related information?

- Interplay between communication and domain-awareness
- Domains useful in both process specifications and type structure
- Enforcing correctness (preservation, progress, termination)

A concurrent interpretation of LL with **hybrid connectives**

- Modal worlds $\mathrm{w}, \mathrm{w}_1, \ldots$ as domains for distributed processes
- At the type level, hybrid connective $@_{\mathrm{w}}$ as session migration
- At the process level, prefixes for domain-tagged channel passing
- Parametric accessibility relation governs movement

Our Proposal: Domain-Aware Sessions

How to enhance session interfaces with domain-related information, in a logically motivated way?

- Interplay between communication and domain-awareness
- Domains useful in both process specifications and type structure
- Enforcing correctness (preservation, progress, termination)

A concurrent interpretation of LL with hybrid connectives

- Modal worlds $\mathrm{w}, \mathrm{w}_1, \ldots$ as domains for distributed processes
- At the type level, hybrid connective $@_{\mathrm{w}}$ as session migration
- At the process level, prefixes for domain-tagged channel passing
- Parametric accessibility relation governs movement

Our Proposal: Domain-Aware Sessions

How to enhance session interfaces with domain-related information, in a logically motivated way?

- Interplay between communication and domain-awareness
- Domains useful in both process specifications and type structure
- Enforcing correctness (preservation, progress, termination)

A concurrent interpretation of LL with hybrid connectives

- Modal worlds $\mathbf{w}, \mathbf{w}_1, \ldots$ as domains for distributed processes
- At the type level, hybrid connective $@_{\mathbf{w}}$ as session migration
- At the process level, prefixes for domain-tagged channel passing
- Parametric accessibility relation governs movement

Domain-Aware Sessions in LL

The perspective of session provider, extended with hybrid type $@_{\mathbf{w}}$:

$c : A \otimes B$	send name $d : A$ on c, continue as B
$c : A \multimap B$	receive name $d : A$ on c, continue as B
$c : \mathbf{1}$	close name c and terminate
$c : \oplus\{\mathtt{l}_i : A_i\}$	send label $\mathtt{l}_i$ on c, continue as A_i
$c : \&\{\mathtt{l}_i : A_i\}$	receive label $\mathtt{l}_i$ on c, continue as A_i
$c : !A$	send persistent $!u : A$ on c and terminate
$!u : A$	receive $c : A$ on $!u$ for fresh instance of A

$c : @_{\mathbf{w}}\, A$	send name $d : A$ on c, continue as A in domain $\mathbf{w}$

Domain-Aware Sessions in LL

The perspective of session provider, extended with hybrid type $@_{\mathbf{w}}$:

$c : A \otimes B$	send name $d : A$ on c, continue as B
$c : A \multimap B$	receive name $d : A$ on c, continue as B
$c : \mathbf{1}$	close name c and terminate
$c : \oplus\{\mathtt{l}_i : A_i\}$	send label $\mathtt{l}_i$ on c, continue as A_i
$c : \&\{\mathtt{l}_i : A_i\}$	receive label $\mathtt{l}_i$ on c, continue as A_i
$c : !A$	send persistent $!u : A$ on c and terminate
$!u : A$	receive $c : A$ on $!u$ for fresh instance of A

$c : @_{\mathbf{w}}\, A$	send name $d : A$ on c, continue as A in domain $\mathbf{w}$

Type Store with Domain Information (1)

We may refine type Store with a reference to trusted domain '$\mathbf{sec}$':

$$\mathsf{Store}_d \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\mathtt{later} : \mathsf{SaveStore} \mathbin{\&} \mathtt{now} : @_{\mathbf{sec}}\, \mathsf{PayStore})$$

Intuitively:

- A migration step to $\mathbf{sec}$ must precede the payment sub-protocol
- Store_d assumed to be located in some domain, say $\mathbf{pub}$. Domain $\mathbf{pub}$ should be entitled to reach domain $\mathbf{sec}$

Two key points:

+ Precision: Migration is well localized within the type interface
− Flexibility: Domain $\mathbf{sec}$ is "hardwired"

Type Store with Domain Information (1)

We may refine type Store with a reference to trusted domain '**sec**':

$$\mathsf{Store}_d \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\mathtt{later} : \mathsf{SaveStore} \,\&\, \mathtt{now} : @_{\mathbf{sec}}\, \mathsf{PayStore})$$

Intuitively:

- A migration step to **sec** must precede the payment sub-protocol
- Store_d assumed to be located in some domain, say **pub**. Domain **pub** should be entitled to reach domain **sec**

Two key points:

+ Precision: Migration is well localized within the type interface
– Flexibility: Domain **sec** is "hardwired"

Type Store with Domain Information (1)

We may refine type Store with a reference to trusted domain '$\mathbf{sec}$':

$$\mathsf{Store}_d \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\mathtt{later} : \mathsf{SaveStore} \,\&\, \mathtt{now} : @_{\mathbf{sec}}\, \mathsf{PayStore})$$

Intuitively:

- A migration step to $\mathbf{sec}$ must precede the payment sub-protocol
- Store_d assumed to be located in some domain, say $\mathbf{pub}$. Domain $\mathbf{pub}$ should be entitled to reach domain $\mathbf{sec}$

Two key points:

+ Precision: Migration is well localized within the type interface
− Flexibility: Domain $\mathbf{sec}$ is "hardwired"

Domain-Aware Sessions in LL

Pass around domains via quantification over worlds:

$c : A \otimes B$	send name $d : A$ on c, continue as B
$c : A \multimap B$	receive name $d : A$ on c, continue as B
$c : \mathbf{1}$	close name c and terminate
$c : \oplus\{\mathtt{l}_i{:}A_i\}$	send label $\mathtt{l}_i$ on c, continue as A_i
$c : \&\{\mathtt{l}_i{:}A_i\}$	receive label $\mathtt{l}_i$ on c, continue as A_i
$c : !A$	send persistent $!u : A$ on c and terminate
$!u : A$	receive $c : A$ on $!u$ for fresh instance of A
$c : @_{\mathbf{w}}\, A$	send name $d : A$ on c, continue as A in domain $\mathbf{w}$
$c : \forall \alpha.\, A$	receive domain $\mathbf{w}$ on c, continue as $A\{\mathbf{w}/\alpha\}$
$c : \exists \alpha.\, A$	send domain $\mathbf{w}$ on c, continue as $A\{\mathbf{w}/\alpha\}$

Domain-Aware Sessions in LL

Pass around domains via quantification over worlds:

$c : A \otimes B$	send name $d : A$ on c, continue as B
$c : A \multimap B$	receive name $d : A$ on c, continue as B
$c : \mathbf{1}$	close name c and terminate
$c : \oplus\{\mathtt{l}_i{:}A_i\}$	send label $\mathtt{l}_i$ on c, continue as A_i
$c : \&\{\mathtt{l}_i{:}A_i\}$	receive label $\mathtt{l}_i$ on c, continue as A_i
$c : !A$	send persistent $!u : A$ on c and terminate
$!u : A$	receive $c : A$ on $!u$ for fresh instance of A
$c : @_{\mathbf{w}}\, A$	send name $d : A$ on c, continue as A in domain $\mathbf{w}$
$c : \forall\alpha.\, A$	receive domain $\mathbf{w}$ on c, continue as $A\{\mathbf{w}/\alpha\}$
$c : \exists\alpha.\, A$	send domain $\mathbf{w}$ on c, continue as $A\{\mathbf{w}/\alpha\}$

Type WStore with Domain Information (2)

We may now define:

$$\mathsf{Store}_{\exists} \triangleq \mathrm{item} \multimap \mathrm{bool} \otimes (\mathtt{later} : \mathsf{SaveStore} \,\&\, \mathtt{now} : \exists \alpha.\, @_{\alpha}\, \mathsf{PayStore})$$

Intuitively:

- Parameter α stands for a domain, reachable from $\mathbf{w}$, but unknown to clients of $\mathsf{Store}_{\exists}$.
- The store process will send a domain reference to the client. Then, coordinated domain migration may follow.

Domain-Aware Session Processes

- A concurrent interpretation of HILL: ILL + modal worlds + $@_{\mathbf{w}}$
- Generalizes the interpretation of Caires and Pfenning:
  - ⋆ Processes extended with prefixes for domain migration:

$$x\langle y@\mathbf{w}\rangle,\ x(y@\mathbf{w}),\ x\langle \mathbf{w}\rangle,\ x(\alpha)$$

  - ⋆ Judgements now stipulate required services AND their domains:

$$\Omega;\ c_1{:}A_1[\mathbf{w}_1], \ldots, c_n{:}A_n[\mathbf{w}_n] \vdash P :: d : C[\mathbf{w}]$$

Well-typed domain-aware session processes

- Respect connectedness relations —communication between unreachable worlds is disallowed.
- Moreover, fidelity, safety, progress, and termination also hold.

Outline

Large-Scale Software Systems: Protocols

- Conveniently described as chroreographies
  - ⋆ A global description of the overall interactive scenario
  - ⋆ Descriptions of the local behavior for each participant
  - ⋆ Ways of checking conformance of local implementations wrt global descriptions. Top-down and bottom-up techniques.
- Several analysis techniques proposed, including:
  - ⋆ Models/standards for (semi)formal description (e.g., BPMN)
  - ⋆ Automata-based approaches (e.g., MSCs/MSGs, CFSMs)
  - ⋆ Type-based approaches, such as session types

Multiparty Session Types

Multiparty Session Types (MPSTs) [Honda, Yoshida, Carbone (2008)]

- Protocols may involve more than two partners
- Global and local types, related by a projection function
- Underlying theory is subtle; analysis techniques hard to obtain

Foundational Significance: Sound and complete characterization though communicating automata. [Deniélou and Yoshida (2013)]

Binary Session Types (BSTs) [Honda, Vasconcelos, Kubo (1998)]

- Protocols involve exactly two partners
- Correctness depends on action compatibility — type duality
- Well-understood theory and analysis techniques

Foundational Significance: Linear logic propositions as session types [Caires & Pfenning (2010); Wadler (2012)]

Multiparty Session Types

Multiparty Session Types (MPSTs) [Honda, Yoshida, Carbone (2008)]

- Protocols may involve more than two partners
- Global and local types, related by a projection function
- Underlying theory is subtle; analysis techniques hard to obtain

Foundational Significance: Sound and complete characterization though communicating automata. [Deniélou and Yoshida (2013)]

Binary Session Types (BSTs) [Honda, Vasconcelos, Kubo (1998)]

- Protocols involve exactly two partners
- Correctness depends on action compatibility — type duality
- Well-understood theory and analysis techniques

Foundational Significance: Linear logic propositions as session types [Caires & Pfenning (2010); Wadler (2012)]

Multiparty and Binary Session Types

Multiparty Session Types (MPSTs) [Honda, Yoshida, Carbone (2008)]

- Protocols may involve more than two partners
- Global and local types, related by a projection function
- Underlying theory is subtle; analysis techniques hard to obtain

Foundational Significance: Sound and complete characterization though communicating automata. [Deniélou and Yoshida (2013)]

Binary Session Types (BSTs) [Honda, Vasconcelos, Kubo (1998)]

- Protocols involve exactly two partners
- Correctness depends on action compatibility — type duality
- Well-understood theory and analysis techniques

Foundational Significance: Linear logic propositions as session types [Caires & Pfenning (2010); Wadler (2012)]

A Commit Protocol as a MPST

A **global description** of the interaction between A, B, and C

$$
\begin{aligned}
G = \mathtt{A} \twoheadrightarrow \mathtt{B}{:}\Big\{ & \mathtt{act}\langle\mathsf{int}\rangle. \\
& \qquad \mathtt{B} \twoheadrightarrow \mathtt{C}{:}\big\{\mathtt{sig}\langle\mathsf{str}\rangle. \\
& \qquad\qquad \mathtt{A} \twoheadrightarrow \mathtt{C}{:}\{\mathtt{com}\langle\mathbf{1}\rangle.\mathsf{end}\}\big\}, \\
& \mathtt{quit}\langle\mathsf{int}\rangle. \\
& \qquad \mathtt{B} \twoheadrightarrow \mathtt{C}{:}\big\{\mathtt{save}\langle\mathbf{1}\rangle. \\
& \qquad\qquad \mathtt{A} \twoheadrightarrow \mathtt{C}{:}\{\mathtt{fin}\langle\mathbf{1}\rangle.\mathsf{end}\}\ \big\}\ \Big\}
\end{aligned}
$$

The **local projections** of global type G onto A and C

$$G\!\restriction\!\mathtt{A} = \mathtt{A}!\big\{\mathtt{act}\langle\mathsf{int}\rangle.\mathtt{A}!\{\mathtt{com}\langle\mathbf{1}\rangle.\mathsf{end}\}, \mathtt{quit}\langle\mathsf{int}\rangle.\mathtt{B}!\{\mathtt{sig}\langle\mathsf{str}\rangle.\mathsf{end}\}\big\}$$

$$G\!\restriction\!\mathtt{C} = \mathtt{B}?\big\{\mathtt{sig}\langle\mathsf{str}\rangle.\mathtt{A}?\{\mathtt{com}\langle\mathbf{1}\rangle.\mathsf{end}\}, \mathtt{save}\langle\mathbf{1}\rangle.\mathtt{A}?\{\mathtt{fin}\langle\mathbf{1}\rangle.\mathsf{end}\}\ \big\}$$

Can MPSTs Be Reduced Into BSTs?

- A reduction would be
  - ⋆ theoretically insightful
  - ⋆ practically useful
- Could we decompose global specifications into binary fragments, preserving sequencing information in interactions?
- Practice suggests that MPSTs are more expressive than BSTs
- Open problem: We don't know of any formal results

Can MPSTs Be Reduced Into BSTs?

- A reduction would be
  - theoretically insightful
  - practically useful
- Could we decompose global specifications into binary fragments, preserving sequencing information in interactions?
- Practice suggests that MPSTs are more expressive than BSTs
- Open problem: We don't know of any formal results

Can MPSTs Be Reduced Into BSTs?

- A reduction would be
  - theoretically insightful
  - practically useful
- Could we decompose global specifications into binary fragments, preserving sequencing information in interactions? – Non trivial!
- Practice suggests that MPSTs are more expressive than BSTs
- Open problem: We don't know of any formal results

Can MPSTs Be Reduced Into BSTs?

- A reduction would be
  - theoretically insightful
  - practically useful
- Could we decompose global specifications into binary fragments, preserving sequencing information in interactions? – Non trivial!
- Practice suggests that MPSTs are more expressive than BSTs
- Open problem: We don't know of any formal results

Can MPSTs Be Reduced Into BSTs?

- A reduction would be
  - theoretically insightful
  - practically useful
- Could we decompose global specifications into binary fragments, preserving sequencing information in interactions? – Non trivial!
- Practice suggests that MPSTs are more expressive than BSTs
- Open problem: We don't know of any formal results

Recent Development: A Positive Result

A formal, two-way correspondence between

- MPSTs with labeled communication and parallel composition, following [Honda, Yoshida, Carbone (2008), Deniélou and Yoshida (2013)]
- BSTs based on linear logic [Caires and Pfenning (2010)]: session fidelity, safety, and progress by typing.

Our Approach: Medium Processes

- We decouple every directed, labeled communication

$$\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{\mathtt{lab}\langle U\rangle.G\}$$

into two actions:

  - ⋆ A send action from $\mathtt{p}$ to some intermediate entity
  - ⋆ A forwarding action from the entity to $\mathtt{q}$

- Given a global type G, extract its medium process $\mathsf{M}[\![G]\!]$
  - ⋆ Intermediate party in all multiparty exchanges
  - ⋆ Captures sequencing information in G by decoupling interactions
  - ⋆ Local implementations need not know about the medium

Our Approach: Medium Processes

- We decouple every directed, labeled communication

$$\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{\mathtt{lab}\langle U\rangle.G\}$$

  into two actions:
  - ⋆ A send action from $\mathtt{p}$ to some intermediate entity
  - ⋆ A forwarding action from the entity to $\mathtt{q}$

- Given a global type G, extract its medium process $\mathsf{M}[\![G]\!]$
  - ⋆ Intermediate party in all multiparty exchanges
  - ⋆ Captures sequencing information in G by decoupling interactions
  - ⋆ Local implementations need not know about the medium

A Formal Correspondence

1. Let G be a well-formed MPST.
   Process $\mathsf{M}[\![G]\!]$ is well-typed under an environment composed of BTSs corresponding to the local projections of G.

2. Given a MPST G, let $\mathsf{M}[\![G]\!]$ be a medium process typed under an environment containing some BSTs.
   Such BSTs precisely correspond to the local projections of G.

Two Worlds Connected by Mediums

- Multiparty interactions now explained from two different angles
- Half-way between two essentially distinct, foundational theories
- Clean justifications, based on linear logic, for MPSTs concepts:
  - ⋆ semantics of global types
  - ⋆ definitions of projection/well-formedness
- Naturally handles name passing, delegation, parallel composition
- Direct connection from choreographies to processes
- Techniques for BSTs applicable on global specifications:
  - ⋆ Deadlock freedom
  - ⋆ Typed behavioral equivalences

Outline

Summary: Logical Foundations for STs

Session types (STs) as **intuitionistic** linear logic propositions

- A theory of linear LRs for session-based concurrency
  - Termination (strong normalization) for concurrent processes
  - Practical significance: enhanced session predictability
- A typed observational equivalence over processes, $\approx$
  - Intuitive definition based on type judgments
  - Clarifies further the relationship between proofs and processes

Two Recent Developments

- Domain-aware STs which rely on hybrid linear logic.
  A generalization of the logic interpretation, based on modal worlds, interpreted as domains. Typeful domain connectedness.
- A formal connection between multiparty and binary STs
  Mediums define a simple characterization of choreographies.

ILL as Session Types: A Reading List

CONCUR'10 – *Session Types as Intuitionistic Linear Propositions*
PPDP'11 – *Dependent Session Types*
TLDI'12 – *Towards Concurrent Type Theory*
FOSSACS'12 – *Session-Typed Encodings of the λ-calculus*
ESOP'12 – *Linear Logical Relations for Sessions*
CSL'12 – *Asynchronous Session-Typed Communication*
ESOP'13 – *Behavioral Polymorphism and Parametricity*
ESOP'13 – *Integrating Functions and Sessions via Monads*
TGC'14 – *Corecursion and Non-Divergence in Sessions*

Curry-Howard Correspondences for Concurrency

Overview and Recent Developments

Jorge A. Pérez

university of groningen

University of Brasilia

July 21, 2015