Session Types for Message-Passing Concurrency

Jorge A. Pérez
University of Groningen, The Netherlands
www.jperez.nl - j.a.perez [[at]] rug.nl

ISR - July 2021
(Part 2, v1.1)

Outline

This Course

A bird's eye view on session types for message-passing concurrency, in two parts:

1. Session types before 2010:
   Motivation, key ideas, essential notions of binary and multiparty session types.
2. Session types after 2010:
   The Curry-Howard correspondence between linear logic and session types (aka "propositions as sessions")

This Course

A bird's eye view on session types for message-passing concurrency, in two parts:

1. Session types before 2010:
   Motivation, key ideas, essential notions of binary and multiparty session types.
2. Session types after 2010:
   The Curry-Howard correspondence between linear logic and session types (aka "propositions as sessions")

My proposal: Part 1 $\rightarrow$ Q&A $\rightarrow$ Break $\rightarrow$ Part 2 $\rightarrow$ Q&A

Keywords and Slogans

Concurrency Theory, Message-Passing, Programming Languages, Verification

Keywords and Slogans

Concurrency Theory, Message-Passing, Programming Languages, Verification

- **Type systems**
  Slogan: Well-typed programs can't go wrong (Milner)

Keywords and Slogans

Concurrency Theory, Message-Passing, Programming Languages, Verification

- **Type systems**
  Slogan: Well-typed programs can't go wrong (Milner)
- **Session types** for communication correctness
  Slogan: **What** and **when** should be sent through a channel

Keywords and Slogans

Concurrency Theory, Message-Passing, Programming Languages, Verification

- **Type systems**
  Slogan: Well-typed programs can't go wrong (Milner)
- **Session types** for communication correctness
  Slogan: **What** and **when** should be sent through a channel
- **Process calculi**
  Slogan: The π-calculus treats **processes** like the λ-calculus treats **functions**
- **Propositions as sessions**

Linear logic propositions	$\leftrightarrow$	session types
Proofs	$\leftrightarrow$	π-calculus processes
Cut elimination	$\leftrightarrow$	process communication

Outline

Propositions as sessions

Linear logic propositions	$\leftrightarrow$	session types
Proofs	$\leftrightarrow$	π-calculus processes
Cut elimination	$\leftrightarrow$	process communication

Developed by Caires & Pfenning for intuitionistic linear logic (2010).
Adapted to classical linear logic by Wadler (2012).

Main Features

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

Propositions as sessions

Linear logic propositions	$\leftrightarrow$	session types
Proofs	$\leftrightarrow$	π-calculus processes
Cut elimination	$\leftrightarrow$	process communication

Developed by Caires & Pfenning for intuitionistic linear logic (2010).
Adapted to classical linear logic by Wadler (2012).

Plan:

- Interpreting propositions in linear logic as session types
- The language of the π-calculus and its semantics
- The sequent calculus as a type system for the π-calculus
- Examples and extensions

Protocols as Session Types

$S ::= \; !\,U; S$		**output** value of type U, continue as S
$\mid \; ?\,U; S$		**input** value of type U, continue as S
$\mid \; \&\{l_i : S_i\}_{i \in I}$		**branching**: offer a selection between $S_1, \ldots, S_n$
$\mid \; \oplus\{l_i : S_i\}_{i \in I}$		**select** one between $S_1, \ldots, S_n$
$\mid \; \mu t.S \mid t$		**recursion**
$\mid \; \mathtt{end}$		**terminated protocol**

(Labels $l_1, \ldots, l_n$ are pairwise different.)

Linear Logic as Session Types (Caires & Pfenning, 2010)

$$
\begin{array}{rll}
A, B ::= & A \otimes B & \text{[Output object of type } A\text{, continue as } B\text{]} \\
\mid & A \multimap B & \text{[Input object of type } A\text{, continue as } B\text{]} \\
\mid & \&\{\mathtt{l}_i : A_i\}_{i \in I} & \text{[Offer } \textit{all of } A_i\text{]} \\
\mid & \oplus\{\mathtt{l}_i : A_i\}_{i \in I} & \text{[Select } \textit{one of } A_i\text{]} \\
\mid & !A & \text{[Persistent offer of } A\text{]} \\
\mid & 1 & \text{[Terminated interaction]}
\end{array}
$$

Linear Logic as Session Types (Caires & Pfenning, 2010)

$$
\begin{array}{rll}
A, B ::= & A \otimes B & \text{[Output object of type } A\text{, continue as } B\text{]} \\
\mid & A \multimap B & \text{[Input object of type } A\text{, continue as } B\text{]} \\
\mid & \&\{\mathtt{l}_i : A_i\}_{i \in I} & \text{[Offer } \textit{all of } A_i\text{]} \\
\mid & \oplus\{\mathtt{l}_i : A_i\}_{i \in I} & \text{[Select } \textit{one of } A_i\text{]} \\
\mid & !A & \text{[Persistent offer of } A\text{]} \\
\mid & 1 & \text{[Terminated interaction]}
\end{array}
$$

Notice:

- The multiplicative conjunction $\otimes$ ('tensor') is given a non-commutative reading
- The exponential '!' ('bang') rather than recursion
- We will have assignments enforcing the use of a name according to some type A

A Synchronous π-calculus

We use $x, y, z, \ldots$ to denote **names** (or **channels**).

$$
\begin{array}{rcll}
P, Q & ::= & x\langle z\rangle.P & \text{send } z \text{ on } x, \text{ proceed as } P \\
 & \mid & x(y).P & \text{receive } z \text{ on } x, \text{ proceed as } P\{z/y\}
\end{array}
$$

A Synchronous π-calculus

We use $x, y, z, \ldots$ to denote **names** (or **channels**).

$$
\begin{array}{rcll}
P, Q & ::= & x\langle z\rangle.P & \text{send } z \text{ on } x, \text{ proceed as } P \\
& \mid & x(y).P & \text{receive } z \text{ on } x, \text{ proceed as } P\{z/y\} \\
& \mid & x.\mathtt{case}(P, Q) & \text{branching: offers a choice at } x \\
& \mid & x.\mathtt{inl}; P & \text{select left at } x, \text{ continue as } P \\
& \mid & x.\mathtt{inr}; P & \text{select right at } x, \text{ continue as } P
\end{array}
$$

A Synchronous π-calculus

We use $x, y, z, \ldots$ to denote **names** (or **channels**).

P, Q	$::=$	$x\langle z\rangle.P$	send z on x, proceed as P
	$\mid$	$x(y).P$	receive z on x, proceed as $P\{z/y\}$
	$\mid$	$x.\mathtt{case}(P, Q)$	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
	$\mid$	$!x(y).P$	replicated server at x

A Synchronous π-calculus

We use $x, y, z, \ldots$ to denote **names** (or **channels**).

P, Q	$::=$	$x\langle z\rangle.P$	send z on x, proceed as P
	$\mid$	$x(y).P$	receive z on x, proceed as $P\{z/y\}$
	$\mid$	$x.\mathtt{case}(P, Q)$	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
	$\mid$	$!x(y).P$	replicated server at x
	$\mid$	$[x \leftrightarrow y]$	forwarder: fuses x and y
	$\mid$	$P \mid Q$	parallel composition
	$\mid$	$(\boldsymbol{\nu} y)P$	name restriction
	$\mid$	0	inaction

A Synchronous π-calculus (n-ary)

We use $x, y, z, \ldots$ to denote **names** (or **channels**).

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

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

Some Simple Processes (1)

An authentication server:

$$SBody(s) = s(user).s \triangleright \{\texttt{email} : P,\ \texttt{app} : Q,\ \texttt{sms} : R\}$$

where

$$P = s(pwd).\overline{s}(y).([y \leftrightarrow code] \mid s(c).0) \qquad \text{(for email-based confirmation)}$$
$$Q = s(pin).\overline{s}(y).([y \leftrightarrow qr] \mid s(n).s(c).0) \qquad \text{(for confirmation with a QR code)}$$
$$R = s(pin).\overline{s}(y).([y \leftrightarrow msg] \mid s(n).0) \qquad \text{(for confirmation with a phone)}$$

A candidate client, $Client(s)$:

$$\overline{s}(u).([u \leftrightarrow \text{`}myUser\text{`}] \mid s \triangleleft \texttt{sms}; \overline{s}(n).([n \leftrightarrow \text{`}666\text{`}] \mid s(m).\overline{s}(c).([c \leftrightarrow \text{`}ok\text{`}] \mid 0)))$$

The whole system:

$$(\boldsymbol{\nu} s)(SBody(s) \mid Client(s))$$

Some Simple Processes (2)

The authentication server (where P, Q, and R are as before):

$$SBody(s) = s(user).s \triangleright \{\texttt{email} : P,\ \texttt{app} : Q,\ \texttt{sms} : R\}$$

The candidate client, $Client(s)$:

$$\overline{s}(u).([u \leftrightarrow \text{`}myUser\text{`}] \mid s \triangleleft \texttt{sms}; \overline{s}(n).([n \leftrightarrow \text{`}666\text{`}] \mid s(m).\overline{s}(c).([c \leftrightarrow \text{`}ok\text{`}] \mid 0)))$$

Some Simple Processes (2)

The authentication server (where P, Q, and R are as before):

$$SBody(s) = s(user).s \triangleright \{\texttt{email} : P,\ \texttt{app} : Q,\ \texttt{sms} : R\}$$

The candidate client, $Client(s)$:

$$\overline{s}(u).([u \leftrightarrow \text{`}myUser\text{'}] \mid s \triangleleft \texttt{sms}; \overline{s}(n).([n \leftrightarrow \text{`}666\text{'}] \mid s(m).\overline{s}(c).([c \leftrightarrow \text{`}ok\text{'}] \mid 0)))$$

The whole system, now as a **client-server** interaction:

$$(\nu u)(!u(s).SBody(s) \mid (\nu y)u\langle y \rangle.Client(y))$$

Intuition: The client will communicate on u to request a copy of the server body using a fresh name y; the server will remain available to other client requests.

Operational Semantics

Reduction determines the behavior of a process on its own:

$$x\langle y\rangle.Q \mid x(z).P \longrightarrow Q \mid P\{y/z\}$$

Operational Semantics

Reduction determines the behavior of a process on its own:

$$x\langle y\rangle.Q \mid x(z).P \longrightarrow Q \mid P\{y/z\}$$
$$x\langle y\rangle.Q \mid !x(z).P \longrightarrow Q \mid P\{y/z\} \mid !x(z).P$$

Operational Semantics

Reduction determines the behavior of a process on its own:

$$
\begin{array}{rcl}
x\langle y\rangle.Q \mid x(z).P & \longrightarrow & Q \mid P\{y/z\} \\
x\langle y\rangle.Q \mid !x(z).P & \longrightarrow & Q \mid P\{y/z\} \mid !x(z).P \\
x.\mathtt{inr}; P \mid x.\mathtt{case}(Q,R) & \longrightarrow & P \mid R \\
x.\mathtt{inl}; P \mid x.\mathtt{case}(Q,R) & \longrightarrow & P \mid Q
\end{array}
$$

Operational Semantics

Reduction determines the behavior of a process on its own:

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

Operational Semantics

Reduction determines the behavior of a process on its own:

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

Closed under structural congruence, noted $\equiv$.

Propositions as Session Types (CONCUR'10)

The assignment $x{:}A$ enforces the use of name x according to type A:

$$
\begin{array}{rcll}
A, B & ::= & A \otimes B & \text{[Output name of type } A\text{, continue as } B\text{]} \\
& | & A \multimap B & \text{[Input name of type } A\text{, continue as } B\text{]} \\
& | & \&\{\mathtt{l}_i : A_i\}_{i \in I} & \text{[Offer } \textit{all of } A_i\text{]} \\
& | & \oplus\{\mathtt{l}_i : A_i\}_{i \in I} & \text{[Select } \textit{one of } A_i\text{]} \\
& | & !A & \text{[Persistent offer of } A\text{]} \\
& | & 1 & \text{[Terminated interaction]}
\end{array}
$$

Type Judgments: Intuitions

$$P :: z : C$$

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

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, $\cdots$, A_n at x_n

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, $\cdots$, A_n at x_n

Examples

$\Delta \vdash$	$P :: z : 1$	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

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

(Names u_i, x_j, z pairwise distinct.)

Intuition: Process P offers behavior C at name z when composed with processes implementing the behaviors described in Γ and Δ.

Type Judgments

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

(Names u_i, x_j, z pairwise distinct.)

Intuition: Process P offers behavior C at name z when composed with processes implementing the behaviors described in Γ and Δ.

Dependencies as two sets of type assignments (contexts), Γ and Δ:

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

Typing Rules

In its intuitionistic formulation, 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

Rules for **cut** in sequent calculus read as well-typed process composition, based on restriction and parallel composition.

Some Typing Rules: Identity, $\otimes$, and &

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

Some Typing Rules: Identity, $\otimes$, and &

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

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

Some Typing Rules: Identity, $\otimes$, and &

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

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

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

Some Typing Rules: Identity, $\otimes$, and &

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

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

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

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

Some Typing Rules: Identity, $\otimes$, and &

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

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

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

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

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

Some Typing Rules: $\multimap$ and $\oplus$

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

Some Typing Rules: $\multimap$ and $\oplus$

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

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta', x : B \vdash Q :: T}{\Gamma; \Delta, \Delta', x : A \multimap B \vdash \overline{x}(y).(P \mid Q) :: T}$$

Some Typing Rules: $\multimap$ and $\oplus$

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

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta', x : B \vdash Q :: T}{\Gamma; \Delta, \Delta', x : A \multimap B \vdash \overline{x}(y).(P \mid Q) :: T}$$

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

Some Typing Rules: $\multimap$ and $\oplus$

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

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta', x : B \vdash Q :: T}{\Gamma; \Delta, \Delta', x : A \multimap B \vdash \overline{x}(y).(P \mid Q) :: T}$$

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

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

Typing Composition

Linear Composition

Cut as composition principle for linear services:

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

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:

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

Linear Cut as Process Reduction

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

$$\longrightarrow$$

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

Shared Cut as Process Reduction

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

$$\longrightarrow$$

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

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.

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 $P \equiv (\boldsymbol{\nu}\tilde{n})(\pi.Q \mid R)$ for some $\pi.Q, R, \tilde{n}$ where $\pi.Q$ is a non-replicated guarded process.

Theorem (Global Progress / Deadlock Avoidance)

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

Also: Termination / Strong Normalization.

Outline

Binary Session Types (BSTs)

- Exactly two partners
- Correctness relies on action compatibility
- Well-understood theory and analysis techniques

Binary Session Types (BSTs)

- Exactly two partners
- Correctness relies on action compatibility
- Well-understood theory and analysis techniques

Multiparty Session Types (MPSTs)

- More than two partners
- Global and local types, related by projection
- Subtle underlying theory; analysis techniques hard to obtain

Binary Session Types (BSTs)

- Exactly two partners
- Correctness relies on action compatibility
- Well-understood theory and analysis techniques

Foundational significance:

Curry-Howard correspondence with linear logic [Caires&Pfenning'10; Wadler'12]

Multiparty Session Types (MPSTs)

- More than two partners
- Global and local types, related by projection
- Subtle underlying theory; analysis techniques hard to obtain

Binary Session Types (BSTs)

- Exactly two partners
- Correctness relies on action compatibility
- Well-understood theory and analysis techniques

Foundational significance:

Curry-Howard correspondence with linear logic [Caires&Pfenning'10; Wadler'12]

Multiparty Session Types (MPSTs)

- More than two partners
- Global and local types, related by projection
- Subtle underlying theory; analysis techniques hard to obtain

Foundational significance:

Characterization via communicating automata (CFSMs)

[Deniélou&Yoshida'12,13; Lange,Tuosto,Yoshida'15]

Can MPSTs Be Reduced Into BSTs?

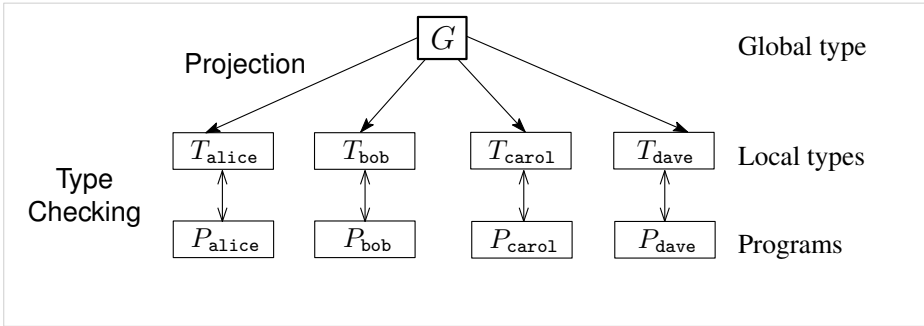
- A reduction would be insightful and practically useful
- Practice suggests MPSTs are more expressive than BSTs
- **Challenge**: Decompose global specs into binary pieces
  - preserving sequencing information
  - avoiding communication errors
  - retaining significance of standard models

A Positive Answer

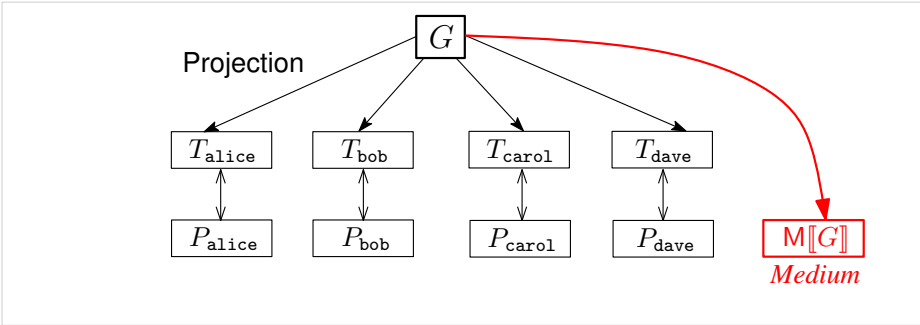
A **two-way correspondence** between

- Standard MPSTs with communication & composition, following [Honda,Yoshida,Carbone'08; Deniélou & Yoshida'13]
- BSTs based on linear logic, following [Caires & Pfenning'10]: fidelity, safety, termination, (dead)lock-freedom by typing

Our Approach: Medium Processes

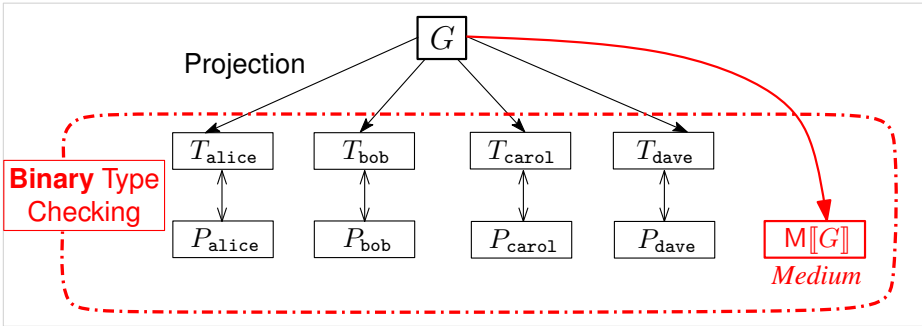


Our Approach: Medium Processes



- The **medium process** $\mathsf{M}[\![G]\!]$
  - Intermediate party in all exchanges in G
  - Captures sequencing information in G by decoupling interactions
- Local implementations need not know about $\mathsf{M}[\![G]\!]$

Our Approach: Medium Processes



- The **medium process** M⟦G⟧
  - Intermediate party in all exchanges in G
  - Captures sequencing information in G by decoupling interactions
- Local implementations need not know about M⟦G⟧

MPSTs: Syntax

- Define global types G and local types T as

$$
\begin{array}{rcl}
G & ::= & \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{\mathtt{l}_i\langle U_i\rangle.G_i\}_{i\in I} \;\mid\; G_1 \mid G_2 \;\mid\; \mathtt{end} \\
T & ::= & \mathtt{p}?\{\mathtt{l}_i\langle U_i\rangle.T_i\}_{i\in I} \;\mid\; \mathtt{p}!\{\mathtt{l}_i\langle U_i\rangle.T_i\}_{i\in I} \;\mid\; \mathtt{end} \\
U & ::= & \mathsf{bool} \mid \mathsf{nat} \mid \mathsf{str} \mid \ldots \mid T
\end{array}
$$

- The global type syntax subsumes those given in [Honda, Yoshida, Carbone'08; Deniélou and Yoshida'13]
- $G \upharpoonright \mathtt{p}_i$ is the **projection** of G onto participant $\mathtt{p}_i$ (merge-based)
- **Well-formedness**: Correct projectability on all participants

Example: A Commit Protocol

Structured interaction among three participants $\mathtt{p}$, $\mathtt{q}$, and $\mathtt{r}$:

$$
\begin{array}{l}
G = \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\Big\{\mathtt{act}\langle \mathsf{int}\rangle.\ \mathtt{q} \twoheadrightarrow \mathtt{r}{:}\Big\{\mathtt{sig}\langle \mathsf{str}\rangle.\ \mathtt{p} \twoheadrightarrow \mathtt{r}{:}\{\mathtt{comm}\langle 1\rangle.\mathtt{end}\}\Big\}, \\
\qquad\qquad \mathtt{quit}\langle \mathsf{int}\rangle.\ \mathtt{q} \twoheadrightarrow \mathtt{r}{:}\Big\{\mathtt{save}\langle 1\rangle.\ \mathtt{p} \twoheadrightarrow \mathtt{r}{:}\{\mathtt{fin}\langle 1\rangle.\mathtt{end}\}\ \Big\}\ \Big\}
\end{array}
$$

The projections of G onto $\mathtt{p}$ and $\mathtt{r}$:

$$
\begin{array}{ll}
G\upharpoonright \mathtt{p} = & \mathtt{q}!\Big\{\mathtt{act}\langle \mathsf{int}\rangle.\mathtt{r}!\{\mathtt{comm}\langle 1\rangle.\mathtt{end}\},\quad \mathtt{quit}\langle \mathsf{int}\rangle.\mathtt{r}!\{\mathtt{fin}\langle 1\rangle.\mathtt{end}\}\Big\} \\
G\upharpoonright \mathtt{q} = & \mathtt{p}?\Big\{\mathtt{act}\langle \mathsf{int}\rangle.\mathtt{r}!\{\mathtt{sig}\langle \mathsf{str}\rangle.\mathtt{end}\},\quad \mathtt{quit}\langle \mathsf{int}\rangle.\mathtt{q}!\{\mathtt{save}\langle 1\rangle.\mathtt{end}\}\Big\} \\
G\upharpoonright \mathtt{r} = & \mathtt{q}?\Big\{\mathtt{sig}\langle \mathsf{str}\rangle.\mathtt{p}?\{\mathtt{comm}\langle 1\rangle.\mathtt{end}\},\quad \mathtt{save}\langle 1\rangle.\mathtt{p}?\{\mathtt{fin}\langle 1\rangle.\mathtt{end}\}\Big\}
\end{array}
$$

Medium Process of a Global Type

- $\mathsf{M}[\![\, \mathtt{p} \to \mathtt{q} : \langle U \rangle.G \,]\!] = c_{\mathtt{p}}(u).\overline{c_{\mathtt{q}}}(v).([u \leftrightarrow v] \mid \mathsf{M}[\![G]\!])$
- $\mathsf{M}[\![\, \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I} \,]\!] = c_{\mathtt{p}} \triangleright \left\{ l_i : c_{\mathtt{q}} \triangleleft l_i ; \mathsf{M}[\![G_i]\!] \right\}_{i \in I}$

Medium Process of a Global Type

- $\mathsf{M}[\![\, \mathtt{p} \to \mathtt{q} : \langle U \rangle.G \,]\!] = c_{\mathtt{p}}(u).\overline{c_{\mathtt{q}}}(v).([u \leftrightarrow v] \mid \mathsf{M}[\![G]\!])$
- $\mathsf{M}[\![\, \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I} \,]\!] = c_{\mathtt{p}} \triangleright \{ l_i : c_{\mathtt{q}} \triangleleft l_i ; \mathsf{M}[\![G_i]\!] \}_{i \in I}$
- $\mathsf{M}[\![\, \mathtt{p} \twoheadrightarrow \mathtt{q} : \{\mathtt{l}_i \langle U_i \rangle.G_i\}_{i \in I} \,]\!] =$

  $c_{\mathtt{p}} \triangleright \{ \mathtt{l}_i : c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft \mathtt{l}_i ; \overline{c_{\mathtt{q}}}(v).([u \leftrightarrow v] \mid \mathsf{M}[\![G_i]\!]) \}_{i \in I}$
- $\mathsf{M}[\![\mathsf{end}]\!] = 0$

An Example: The Commit Protocol

$$G = \mathtt{p} \twoheadrightarrow \mathtt{q}:\Big\{\mathtt{act}\langle \mathsf{int}\rangle.\mathtt{q} \twoheadrightarrow \mathtt{r}:\Big\{\mathtt{sig}\langle \mathsf{str}\rangle.\mathtt{p} \twoheadrightarrow \mathtt{r}:\{\mathtt{comm}\langle 1\rangle.\mathtt{end}\}\Big\},$$
$$\mathtt{quit}\langle \mathsf{int}\rangle.\mathtt{q} \twoheadrightarrow \mathtt{r}:\{\mathtt{save}\langle 1\rangle.\mathtt{p} \twoheadrightarrow \mathtt{r}:\{\mathtt{fin}\langle 1\rangle.\mathtt{end}\}\Big\} \quad \Big\}$$

An Example: The Commit Protocol

$$G = \mathtt{p} \twoheadrightarrow \mathtt{q}:\Big\{\mathtt{act}\langle \mathsf{int}\rangle.\mathtt{q} \twoheadrightarrow \mathtt{r}:\Big\{\mathtt{sig}\langle \mathsf{str}\rangle.\mathtt{p} \twoheadrightarrow \mathtt{r}:\{\mathtt{comm}\langle 1\rangle.\mathtt{end}\}\Big\},$$
$$\mathtt{quit}\langle \mathsf{int}\rangle.\mathtt{q} \twoheadrightarrow \mathtt{r}:\{\mathtt{save}\langle 1\rangle.\mathtt{p} \twoheadrightarrow \mathtt{r}:\{\mathtt{fin}\langle 1\rangle.\mathtt{end}\}\Big\} \quad \Big\}$$

- The medium process $\mathsf{M}[\![G]\!]$:

$$c_{\mathtt{p}} \triangleright \Big\{ \mathtt{act} : c_{\mathtt{p}}(v).c_{\mathtt{q}} \triangleleft \mathtt{act}; \overline{c_{\mathtt{q}}}(w).([w \leftrightarrow v] \mid$$
$$c_{\mathtt{q}} \triangleright \{\mathtt{sig} : c_{\mathtt{q}}(n).c_{\mathtt{r}} \triangleleft \mathtt{sig}; \overline{c_{\mathtt{r}}}(m).([n \leftrightarrow m] \mid$$
$$c_{\mathtt{p}} \triangleright \{\mathtt{comm} : c_{\mathtt{p}}(u).c_{\mathtt{r}} \triangleleft \mathtt{comm}; \overline{c_{\mathtt{r}}}(y).([u \leftrightarrow y] \mid 0)\}\ )\})\ ,$$
$$\mathtt{quit} : c_{\mathtt{p}}(v).c_{\mathtt{q}} \triangleleft \mathtt{quit}; \overline{c_{\mathtt{q}}}(w).([w \leftrightarrow v] \mid$$
$$c_{\mathtt{q}} \triangleright \{\mathtt{save} : c_{\mathtt{q}}(n).c_{\mathtt{r}} \triangleleft \mathtt{save}; \overline{c_{\mathtt{r}}}(m).([n \leftrightarrow m] \mid$$
$$c_{\mathtt{p}} \triangleright \{\mathtt{fin} : c_{\mathtt{p}}(u).c_{\mathtt{r}} \triangleleft \mathtt{fin}; \overline{c_{\mathtt{r}}}(y).([u \leftrightarrow y] \mid 0)\}\ )\}\ )\Big\}$$

MPSTs and BSTs: Correspondence (1/2)

- The type judgment from [Caires & Pfenning'10]:

$$\Gamma; \Delta \vdash P :: x : C$$

P provides behavior C at channel x using "services" in $\Gamma; \Delta$

MPSTs and BSTs: Correspondence (1/2)

- The type judgment from [Caires & Pfenning'10]:

$$\Gamma; \Delta \vdash P :: x : C$$

  P provides behavior C at channel x using "services" in $\Gamma; \Delta$
- A compositional typing gives a binary type for all participants
- Mapping $\langle\!\langle \cdot \rangle\!\rangle$ from local types T to binary session types A

Theorem (Well-Formed $G \rightarrow$ Well Typed $\mathsf{M}[\![G]\!]$)

Let G be a **well-formed** *global type, with* $\mathsf{part}(G) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$. *Then*

$$\Gamma;\, c_{\mathtt{p}_1} : \langle\!\langle G \upharpoonright \mathtt{p}_1 \rangle\!\rangle, \ldots, c_{\mathtt{p}_n} : \langle\!\langle G \upharpoonright \mathtt{p}_n \rangle\!\rangle \vdash \mathsf{M}[\![G]\!] :: -:1$$

is a compositional typing for $\mathsf{M}[\![G]\!]$, *for some* Γ.

MPSTs and BSTs: Correspondence (2/2)

- The type judgment from [Caires & Pfenning'10]:

$$\Gamma; \Delta \vdash P :: x : C$$

- A compositional typing gives a binary type for all participants.
- Mapping $\langle\!\langle \cdot \rangle\!\rangle$ from local types T to binary session types A
- Ordering $\preceq^{\sqcup}$ relates local branching types (akin to subtyping)

Theorem (Well-Typedness → WF Global Types)

Let G be a global type. If

$$\Gamma; c_{\mathtt{p}_1}{:}A_1, \ldots, c_{\mathtt{p}_n}{:}A_n \vdash \mathsf{M}[\![G]\!] :: -{:}1$$

is a compositional typing for $\mathsf{M}[\![G]\!]$ then there are local types $T_1, \ldots, T_n$ s.t. $G\upharpoonright \mathtt{r}_j \preceq^{\sqcup} T_j$ and $\langle\!\langle T_j \rangle\!\rangle = A_j$, for all $\mathtt{r}_j \in G$.

Operational Correspondence

Medium processes faithfully mirror global types

- The annotated medium $\mathcal{M}[\![G]\!]_k$ uses a fresh k to mimic each action of G.
- If G is well-formed then we have, for some Γ:

$$\Gamma;\, c_{\mathrm{p}_1}{:}\langle\!\langle G\!\upharpoonright\!\mathrm{p}_1\rangle\!\rangle, \ldots, c_{\mathrm{p}_n}{:}\langle\!\langle G\!\upharpoonright\!\mathrm{p}_n\rangle\!\rangle \vdash \mathcal{M}[\![G]\!]_k :: k : (\!|G|\!)$$

  $(\!|G|\!)$ is a binary type that captures the sequentiality in G.

Operational Correspondence

Medium processes faithfully mirror global types

- The annotated medium $\mathcal{M}[\![G]\!]_k$ uses a fresh k to mimic each action of G.
- If G is well-formed then we have, for some Γ:

$$\Gamma;\, c_{\mathrm{p}_1}{:}\langle\!\langle G\!\upharpoonright\!\mathrm{p}_1\rangle\!\rangle, \ldots, c_{\mathrm{p}_n}{:}\langle\!\langle G\!\upharpoonright\!\mathrm{p}_n\rangle\!\rangle \vdash \mathcal{M}[\![G]\!]_k :: k : (\!|G|\!)$$

  $(\!|G|\!)$ is a binary type that captures the sequentiality in G.

Operational correspondence for multiparty systems

- Let $S = (\boldsymbol{\nu}\tilde{c})(P_1 \mid \cdots \mid P_n \mid \mathcal{M}[\![G]\!]_k)$ be a **system** realizing G.
- Every move of G can be mimicked by an action of S on k.

The Commit Protocol, Revisited

Given the commit protocol G, we look at the properties of $\mathsf{M}[\![G]\!]$.

- The bidirectional correspondence ensures

$$\Gamma;\ c_{\mathtt{p}} : \mathsf{P},\ c_{\mathtt{q}} : \mathsf{Q},\ c_{\mathtt{r}} : \mathsf{R} \vdash \mathsf{M}[\![G]\!] :: -:1$$

  for some Γ, with types $\mathsf{P} = \langle\!\langle G \upharpoonright \mathtt{p} \rangle\!\rangle$, $\mathsf{Q} = \langle\!\langle G \upharpoonright \mathtt{q} \rangle\!\rangle$, and $\mathsf{R} = \langle\!\langle G \upharpoonright \mathtt{r} \rangle\!\rangle$.

The Commit Protocol, Revisited

Given the commit protocol G, we look at the properties of $\mathsf{M}[\![G]\!]$.

- The bidirectional correspondence ensures

$$\Gamma;\ c_{\mathtt{p}} : \mathsf{P},\ \ c_{\mathtt{q}} : \mathsf{Q},\ \ c_{\mathtt{r}} : \mathsf{R} \vdash \mathsf{M}[\![G]\!] :: -{:}1$$

  for some Γ, with types $\mathsf{P} = \langle\!\langle G{\upharpoonright}\mathtt{p} \rangle\!\rangle$, $\mathsf{Q} = \langle\!\langle G{\upharpoonright}\mathtt{q} \rangle\!\rangle$, and $\mathsf{R} = \langle\!\langle G{\upharpoonright}\mathtt{r} \rangle\!\rangle$.

- We may then just compose $\mathsf{M}[\![G]\!]$ with compatible implementations: $\cdot;\cdot \vdash \mathit{Imp}_{\mathtt{p}} :: c_{\mathtt{p}} : \mathsf{P}$, $\cdot;\cdot \vdash \mathit{Imp}_{\mathtt{q}} :: c_{\mathtt{p}} : \mathsf{Q}$, and $\cdot;\cdot \vdash \mathit{Imp}_{\mathtt{r}} :: c_{\mathtt{p}} : \mathsf{R}$:

$$\Gamma;\cdot \vdash (\boldsymbol{\nu}\, c_{\mathtt{p}})(\mathit{Imp}_{\mathtt{p}} \,|\, (\boldsymbol{\nu}\, c_{\mathtt{q}})(\mathit{Imp}_{\mathtt{q}} \mid (\boldsymbol{\nu}\, c_{\mathtt{r}})(\mathit{Imp}_{\mathtt{r}} \,|\, \mathsf{M}[\![G]\!])))$$

The Commit Protocol, Revisited

Given the commit protocol G, we look at the properties of $\mathsf{M}[\![G]\!]$.

- The bidirectional correspondence ensures

$$\Gamma;\ c_{\mathtt{p}} : \mathsf{P},\ \ c_{\mathtt{q}} : \mathsf{Q},\ \ c_{\mathtt{r}} : \mathsf{R} \vdash \mathsf{M}[\![G]\!] :: -{:}1$$

  for some Γ, with types $\mathsf{P} = \langle\!\langle G \upharpoonright \mathtt{p} \rangle\!\rangle$, $\mathsf{Q} = \langle\!\langle G \upharpoonright \mathtt{q} \rangle\!\rangle$, and $\mathsf{R} = \langle\!\langle G \upharpoonright \mathtt{r} \rangle\!\rangle$.

- We may then just compose $\mathsf{M}[\![G]\!]$ with compatible implementations: $\cdot;\cdot \vdash \mathit{Imp}_{\mathtt{p}} :: c_{\mathtt{p}} : \mathsf{P}$, $\cdot;\cdot \vdash \mathit{Imp}_{\mathtt{q}} :: c_{\mathtt{p}} : \mathsf{Q}$, and $\cdot;\cdot \vdash \mathit{Imp}_{\mathtt{r}} :: c_{\mathtt{p}} : \mathsf{R}$:

$$\Gamma;\cdot \vdash (\boldsymbol{\nu}\, c_{\mathtt{p}})(\mathit{Imp}_{\mathtt{p}} \,|\, (\boldsymbol{\nu}\, c_{\mathtt{q}})(\mathit{Imp}_{\mathtt{q}} \mid (\boldsymbol{\nu}\, c_{\mathtt{r}})(\mathit{Imp}_{\mathtt{r}} \,|\, \mathsf{M}[\![G]\!])))$$

- We derive fidelity, deadlock-freedom, termination from BSTs. Operational correspondence ensures that $\mathsf{M}[\![G]\!]$ is just a router.

Sharing in Multiparty Conversations

- Suppose that implementations to be composed with $\mathsf{M}[\![G]\!]$ must be invoked from a replicated server. For instance: $\cdot;\cdot \vdash !u_1(c_{\mathrm{p}}).\mathit{Imp}_{\mathrm{p}} :: u_1 : !\mathsf{P}$
- We need an "initiator" to spawn a copy of the medium's required type, ready at an appropriate name. For instance:

$$\cdot; u_2 : !\mathsf{Q} \vdash \overline{u_2}(x).[x \leftrightarrow c_{\mathrm{q}}] :: c_{\mathrm{q}} : \mathsf{Q}$$

- Let $\mathit{RepImp}_{\mathrm{p}}$, $\mathit{RepImp}_{\mathrm{q}}$, and $\mathit{RepImp}_{\mathrm{r}}$, denote the composition of replicated definitions and initiators above. Then:

$$\Gamma;\cdot \vdash (\boldsymbol{\nu}\, c_{\mathrm{p}})(\mathit{RepImp}_{\mathrm{p}} \,|\, (\boldsymbol{\nu}\, c_{\mathrm{q}})(\mathit{RepImp}_{\mathrm{q}} \mid (\boldsymbol{\nu}\, c_{\mathrm{r}})(\mathit{RepImp}_{\mathrm{r}} \,|\, \mathsf{M}[\![G]\!])))$$

- The medium could spawn the local implementations itself:

$$\overline{u_1}(c_{\mathrm{p}}).\overline{u_2}(c_{\mathrm{q}}).\overline{u_3}(c_{\mathrm{r}}).\mathsf{M}[\![G]\!]$$

We can show that alternative settings are **bisimilar**

Different Worlds, Linked by Mediums

- MPSTs explained from different angles
- Logic justifications for MPSTs notions:
  - projection, type well-formedness
  - semantics of global types
  - behavioral equivalences (global swapping)
- Connects standard MPSTs to process implementations
- Supports name passing, **delegation**, composition, infinite behavior/sharing
- Techniques for BSTs **applied** to MPSTs
  - deadlock freedom
  - typed behavioral equivalences
  - parametric polymorphism

Outline

Summary

We have overviewed:

- Key notions underlying **binary session types** and **multiparty session types** (without committing to a process model)
- A concurrent interpretation of linear logic that
  - Clarifies the logical foundations of binary session types, in the spirit of the **Curry-Howard isomorphism**
  - Identifies a class of π-calculus processes which enjoy fidelity, safety, and progress
  - Offers a canonical perspective also for multiparty session types

Further Topics

Research on session types has long addressed topics not mentioned here, including:

- Different **liveness properties** (progress, deadlock-freedom, and lock-freedom)
- Synchronous / asynchronous **communication disciplines**
- Connections between session types and **automata theory**
- **Security properties** (secure information flow, access control)
- Integration of session types into **object-oriented, functional, and imperative calculi** and languages
- **Behavioral equivalences** as informed by session types
- Session types and models of **exceptions**, **reversibility**, **run-time monitoring** and **adaptation**

Essential References

- Kohei Honda, Vasco Thudichum Vasconcelos, Makoto Kubo:
  Language Primitives and Type Discipline for Structured Communication-Based Programming. ESOP 1998.
- Kohei Honda, Nobuko Yoshida, Marco Carbone:
  Multiparty asynchronous session types. POPL 2008.
  Also: Journal of the ACM, Volume 63(1): 9 (2016)
- Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, Nobuko Yoshida:
  A Gentle Introduction to Multiparty Asynchronous Session Types. SFM 2015.
- Luís Caires, Frank Pfenning, Bernardo Toninho:
  Linear logic propositions as session types.
  Math. Structures in Comp. Science 26(3): 367-423 (2016)
  (Extended version of a CONCUR 2010 paper.)

Further (Recent) References

- Hans Hüttel et al:
  Foundations of Session Types and Behavioural Contracts. ACM Comput. Surv. 49(1): 3 (2016)
- Davide Ancona et al:
  Behavioral Types in Programming Languages. Foundations and Trends in Programming Languages 3(2-3): 95-230 (2016)
- Luís Caires and Jorge A. Pérez:
  Multiparty Session Types Within a Canonical Binary Theory, and Beyond. FORTE 2016.

Outline

My Group's Current Challenge

- Many behavioral type systems!
- Correctness via various **behavioral properties**
  - Protocol fidelity, comm. safety, deadlock-freedom
- Different type systems, properties and insights

My Group's Current Challenge

- Many behavioral type systems!
- Correctness via various **behavioral properties**
  - Protocol fidelity, comm. safety, deadlock-freedom
- Different type systems, properties and insights
- A program can be both correct *and* incorrect!

Relative Expressiveness

Connect behavioral type systems
by relating the **concurrent languages** on which they operate

Relative Expressiveness

Connect behavioral type systems
by relating the **concurrent languages** on which they operate

Goals

- ✓ **Encodability result**:
  A correct compiler between two concurrent languages
- ✗ **Separation result**:
  A proof that a correct compiler does not exist

Relative Expressiveness

Connect behavioral type systems
by relating the **concurrent languages** on which they operate

Goals

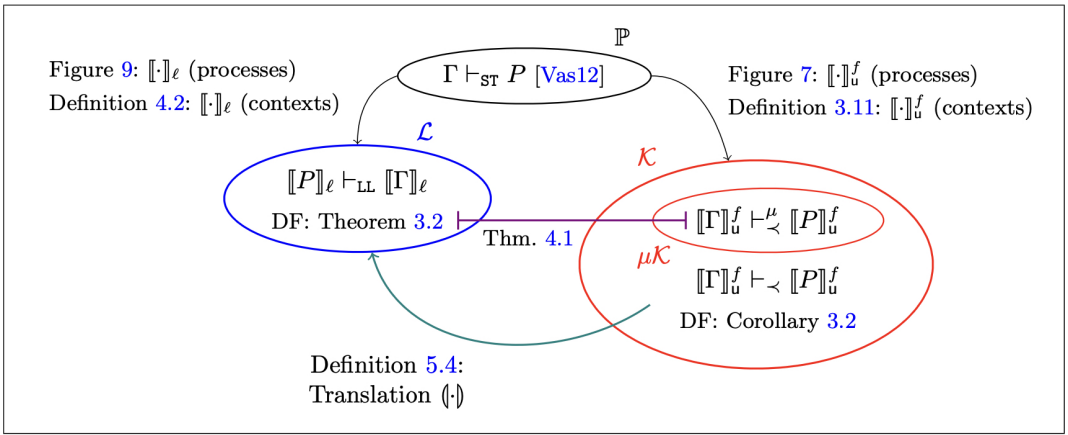
- ✓ **Encodability result**:
  A correct compiler between two concurrent languages
- ✗ **Separation result**:
  A proof that a correct compiler does not exist

Highlights:

⇒ A **general**, **rigorous**, **flexible**, and **practical** approach

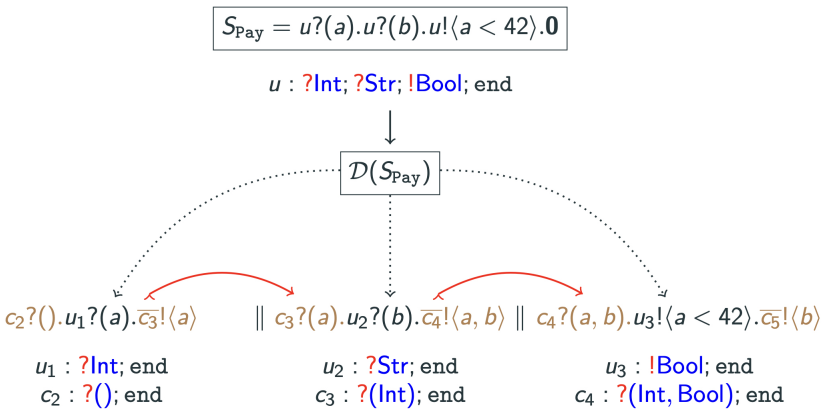
A Recent Result

Classifying three different session type systems for deadlock-freedom (joint work with Ornela Dardha, U of Glasgow):



Another Recent Result

Understanding sequentiality in session types
(joint work Alen Arslanagic and Erik Voogd):



Session Types for Message-Passing Concurrency

Jorge A. Pérez
University of Groningen, The Netherlands
www.jperez.nl - j.a.perez [[at]] rug.nl

ISR - July 2021
(Part 2, v1.1)