Type-based Communication Correctness in Multi-agent Systems

Part II: Type Systems for Concurrency and Logical Foundations

Jorge A. Pérez

Bernoulli Institute for Mathematics, Computer Science, and AI
University of Groningen, The Netherlands

www.jperez.nl

20th European Agent Systems Summer School (EASSS 2018)

Outline

The Future (According to Gartner)

Communication and distribution at a (very) large-scale:

- 2018: 6 billion connected ‘things’ requesting support
- 2020: Autonomous agents part of 5% of all transactions
- 2020: Smart agents facilitate 40% of mobile interactions

The Present: Languages Promoted by Industry

- Facebook's Flow (gradual types for JavaScript)
- Google's Go (concurrency, message-passing communication)
- Mozilla's Rust (affine references/ownership types)
- Erlang (actor-based concurrency)

The Future (According to Gartner)

Communication and distribution at a (very) large-scale:

- 2018: 6 billion connected 'things' requesting support
- 2020: Autonomous agents part of 5% of all transactions
- 2020: Smart agents facilitate 40% of mobile interactions

Communication & Types: **Here to Stay!**

The Present: Languages Promoted by Industry

- Facebook's Flow (gradual types for JavaScript)
- Google's Go (concurrency, message-passing communication)
- Mozilla's Rust (affine references/ownership types)
- Erlang (actor-based concurrency)

The Future (According to Gartner)

Communication and distribution at a (very) large-scale:

- 2018: 6 billion connected 'things' requesting support
- 2020: Autonomous agents part of 5% of all transactions
- 2020: Smart agents facilitate 40% of mobile interactions

Large-scale Software Infrastructures

- Large 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!

Where Do Errors Come From?

Leesatapornwongsa et al. (ASPLOS'16):

TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems

A study of 104 distributed concurrency (DC) bugs from widely-deployed cloud-scale datacenter distributed systems.

Where Do Errors Come From?

Leesatapornwongsa et al. (ASPLOS'16):

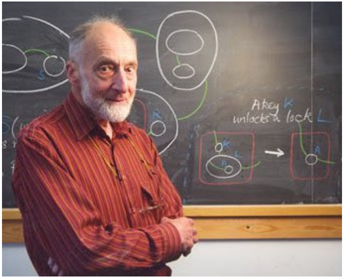
TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems

A study of 104 distributed concurrency (DC) bugs from widely-deployed cloud-scale datacenter distributed systems.

From their summary of findings:

- DC bugs linger in **concurrent executions of multiple protocols**. Systems contain many background protocols beyond user-facing foreground protocols. Their concurrent interactions can be deadly.
- DC bugs triggered by a single untimely message delivery that commits **order violation** or **atomicity violation**, with regard to other messages or computation.

Type Systems: Two Slogans



Robin Milner
ACM Turing Winner, 1991

- Types are the leaven of computer programming: they make it digestible.
- Well-typed programs can't go wrong

Type Systems

Traditional **data types** (e.g., `int`, `bool`, `string`) classify **values**, and are an effective basis for validating sequential programs

Type Systems

Traditional **data types** (e.g., `int`, `bool`, `string`) classify **values**, and are an effective basis for validating sequential programs

To reason about services, **behavioral types** classify **interactions**

- High-level representations of communication structures
- Compositional ways of (statically) checking service behavior
- Tied to programming abstractions that promote communication as a first-class concern

Outline

Type Systems for Concurrency

The development of process languages with type-based techniques has received much attention

Type systems have revealed a **rich landscape** of concurrent models with disciplined communication

Behavioral Type Systems

- In contrast to usual data types, **behavioral types** represent causality, alternatives, repetition.
- Given a communication device (say, a channel), a behavioral type defines
  - the series of actions realized through that device along time
  - its resource-usage policy
- Often developed on top of process calculi, such as the π-calculus.
- General verification techniques that may be tailored to different actual languages:
  - Object-oriented: Java, Scala
  - Functional: Haskell, OCaML
  - Protocol languages: Scribble
- A notable class of behavioral types: **session types**

Behavioral Types: An Incomplete Timeline

- 1990 — **Sortings for the π-calculus** [Milner]
- 1991 — Type & Effect System for CML [Nielson & Nielson]
- 1993 — **Types for Dyadic Interaction** [Honda]
- 1994 — A Typed Interaction-based Language [Takeuchi et al.]
- 1996 — Linear Types for π [Kobayashi et al.]
- 1997 — Types for Deadlock-freedom in π [Kobayashi]
- 1998 — **Binary Session Types** [Honda et al.]

2017

Behavioral Types: An Incomplete Timeline

1991	**Sortings for the π-calculus**	[Milner]
1993	Type & Effect System for CML	[Nielson & Nielson]
1993	**Types for Dyadic Interaction**	[Honda]
1994	A Typed Interaction-based Language	[Takeuchi et al.]
1996	Linear Types for π	[Kobayashi et al.]
1997	Types for Deadlock-freedom in π	[Kobayashi]
1998	**Binary Session Types**	[Honda et al.]
2000	Types for Non-Uniform Objects	[Ravara & Vasconcelos]
2001	Generic Process Types	[Igarashi & Kobayashi]
2007	Multipoint Session Types	[Bonelli & Compagnoni]
2008	**Multiparty Session Types**	[Honda et al.]
2009	Conversation Types	[Caires & Vieira]
2010	Parameterized Multiparty Session Types	[Yoshida et al.]
2010	Linear Session Types, Revisited	[Giunti & Vasconcelos]
2010	**Session Types as Linear Logic**	[Caires & Pfenning]
2011	Dynamic Multirole Session Types	[Deniélou & Yoshida]
2013	Choreographic Programming	[Carbone & Montesi]
2013	Behavioral Separation Types	[Caires & Seco]

Outline

Session-Based Concurrency

Conceptually, two phases:

1. Services advertise their session protocols along **channel names**.
   Agreements are realized by their point-to-point interaction, in an **unrestricted** and **non-deterministic** way.
2. After agreement, compatible services establish a unique session along (fresh, private) **session names**.
   Intra-session interactions follow the intended protocol in a **linear** and **deterministic** way.

The Language of Session Types

Session types describe protocols in terms of

- communication actions (input and output)
- labeled choices (offers and selections)
- sequential composition
- recursion

Session protocols are associated to **communication devices**:

- π-calculus names
- service endpoints
- TCP-IP sockets
- $\cdots$

The Syntax of Binary Session Types

$S ::=$	$!U.S$	**output** value of type U, continue as S
$\mid$	$?U.S$	**input** value of type U, continue as S
$\mid$	$\&\{l_i : S_i\}_{i \in I}$	**offer** a selection between $S_1, \ldots, S_n$ labels $l_1, \ldots, l_n$ are pairwise different
$\mid$	$\oplus\{l_i : S_i\}_{i \in I}$	**select** between $S_1, \ldots, S_n$ labels $l_1, \ldots, l_n$ are pairwise different
$\mid$	$\mu t.S \mid t$	**recursion**
$\mid$	end	**terminated protocol**

Notice:

- The syntax of U refers to "basic values" (e.g. `int`, `bool`, ...) but it may also could contain S — aka session delegation
- Sequential communication patterns (no built-in concurrency)

Example: A Two-Buyer Protocol

Alice and Bob cooperate in buying a book from Seller.

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks with Bob whether he can contribute in buying the book.
3. Alice uses the answer from Bob to interact with Seller, either
   a) completing the payment and arranging delivery details
   b) canceling the transaction
4. In case 3(a) Alice contacts Bob to get his address, and forwards it to Seller.

4'. In case 3(b) Alice is in charge of gracefully concluding the conversation.

Example: A Two-Buyer Protocol

Desiderata for the implementations of Alice, Bob, and Seller:

- **Fidelity** – they **follow the intended protocol.** For instance:
  - Alice doesn't continue the transaction if Bob can't contribute
  - Alice chooses among the options provided by Seller
- **Safety** – they don't feature **communication errors.**
  For instance: Seller always returns an integer when asked by Alice to provide a quote
- **Progress/Deadlock-Freedom** – they do not **"get stuck"** while running the protocol.
  For instance: Alice eventually receives an answer from Bob on his contribution to the transaction.
- **Termination** – they do not engage in **infinite behavior** (that may prevent them from completing the protocol)

Example: A Two-Buyer Protocol

Desiderata for the implementations of Alice, Bob, and Seller:

- **Fidelity** – they **follow the intended protocol**. For instance:
  - Alice doesn't continue the transaction if Bob can't contribute
  - Alice chooses among the options provided by Seller
- **Safety** – they don't feature **communication errors**.
  For instance: Seller always returns an integer when asked by Alice to provide a quote
- **Progress/Deadlock-Freedom** – they do not **"get stuck"** while running the protocol.
  For instance: Alice eventually receives an answer from Bob on his contribution to the transaction.
- **Termination** – they do not engage in **infinite behavior** (that may prevent them from completing the protocol)

Correctness follows from their interplay. This is **hard to enforce**, especially if actions are "scattered around" in source programs.

Example: A Two-Buyer Protocol

We may define two separate protocols, with Alice "leading" the interactions (later on we will consider a simpler solution):

- A session type for Seller (in its interaction with Alice):

$$S_1 = ?\mathtt{book}.\,!\mathtt{quote}.\ \& \begin{cases} \mathsf{buy}: & ?\mathtt{paym}.\,?\mathtt{address}.\,!\mathtt{ok}.\,\mathsf{end} \\ \mathsf{cancel}: & ?\mathtt{thanks}.\,!\mathtt{bye}.\,\mathsf{end} \end{cases}$$

- A session type for Alice (in its interaction with Bob):

$$S_2 = !\mathtt{cost}.\ \& \begin{cases} \mathsf{share}: & ?\mathtt{address}.\,!\mathtt{ok}.\,\mathsf{end} \\ \mathsf{close}: & !\mathtt{bye}.\,\mathsf{end} \end{cases}$$

Example: A Two-Buyer Protocol

Implementations for Alice, Bob, Seller should be **compatible**.

Example: A Two-Buyer Protocol

Implementations for Alice, Bob, Seller should be **compatible**.

- Using session types, compatibility follows from **type duality**, which relates types with opposite behaviors. Intuitively:
  - the dual of input is output (and vice versa)
  - branching is the dual of selection (and vice versa)

Example: A Two-Buyer Protocol

Implementations for Alice, Bob, Seller should be **compatible**.

- Using session types, compatibility follows from **type duality**, which relates types with opposite behaviors. Intuitively:
  - the dual of input is output (and vice versa)
  - branching is the dual of selection (and vice versa)

- This way, e.g., the implementation of Bob should conform to the dual of S_2, denoted $\overline{S_2}$:

$$S_2 = \ !\mathtt{cost}.\ \& \begin{cases} \text{share}: & ?\mathtt{address}.!\mathtt{ok}.\,\text{end} \\ \text{close}: & !\mathtt{bye}.\,\text{end} \end{cases}$$

$$\overline{S_2} = \ ?\mathtt{cost}.\oplus \begin{cases} \text{share}: & !\mathtt{address}.?\mathtt{ok}.\,\text{end} \\ \text{close}: & ?\mathtt{bye}.\,\text{end} \end{cases}$$

- Also, Alice's implementation should conform to both $\overline{S_1}$ and S_2.

Session Type Duality, Formally

Given a (finite) session type S, its dual type $\overline{S}$ is inductively defined as follows:

$$
\begin{aligned}
\overline{!\,U.S} &= ?\,U.\overline{S} \\
\overline{?\,U.S} &= !\,U.\overline{S} \\
\overline{\&\{l_i : S_i\}_{i\in I}} &= \oplus\{l_i : \overline{S_i}\}_{i\in I} \\
\overline{\oplus\{l_i : S_i\}_{i\in I}} &= \&\{l_i : \overline{S_i}\}_{i\in I} \\
\overline{\mathsf{end}} &= \mathsf{end}
\end{aligned}
$$

Notice:

- Duality for recursive session types is defined coinductively rather than inductively (i.e., the dual of $\mu t.S$ is not just $\mu t.\overline{S}$)

Enhancing Compatibility via Subtyping

Consider a "mathematical server" and two candidate clients.

- The session type for the server:

$$S = \&\begin{cases} \mathsf{add}: & ?\texttt{Real}.?\texttt{Real}.!\texttt{Real}.\mathsf{end} \\ \mathsf{eq}: & ?\texttt{Real}.?\texttt{Real}.!\texttt{Bool}.\mathsf{end} \end{cases}$$

- The session types for each of the clients:

$$\text{Integer client} \quad T_1 = \oplus\begin{cases} \mathsf{add}: & !\texttt{Real}.!\texttt{Real}.?\texttt{Real}.\mathsf{end} \\ \mathsf{eq}: & !\texttt{Int}.!\texttt{Int}.?\texttt{Bool}.\mathsf{end} \end{cases}$$

$$\text{Minimal client} \quad T_2 = \oplus\left\{\mathsf{add}: \ !\texttt{Real}.!\texttt{Real}.?\texttt{Real}.\mathsf{end}\right.$$

Enhancing Compatibility via Subtyping

Consider a "mathematical server" and two candidate clients.

- The session type for the server:

$$S = \&\begin{cases}\textsf{add}: & ?\texttt{Real}.?\texttt{Real}.!\texttt{Real}.\textsf{end}\\ \textsf{eq}: & ?\texttt{Real}.?\texttt{Real}.!\texttt{Bool}.\textsf{end}\end{cases}$$

- The session types for each of the clients:

$$\text{Integer client}\quad T_1 = \oplus\begin{cases}\textsf{add}: & !\texttt{Real}.!\texttt{Real}.?\texttt{Real}.\textsf{end}\\ \textsf{eq}: & !\texttt{Int}.!\texttt{Int}.?\texttt{Bool}.\textsf{end}\end{cases}$$

$$\text{Minimal client}\quad T_2 = \oplus\begin{cases}\textsf{add}: & !\texttt{Real}.!\texttt{Real}.?\texttt{Real}.\textsf{end}\end{cases}$$

- The types are **incompatible**: S and T_1 consider messages of different base types, and the options of S and T_2 do not match.
- Still, the types are "morally" compatible...

Enhancing Compatibility via Subtyping

We may relate S with T_1 and T_2, using a **subtyping** relation.

Enhancing Compatibility via Subtyping

We may relate S with T_1 and T_2, using a **subtyping** relation.

- Notation: $S_1 \leq S_2$ (read: S_1 is a subtype of S_2)
- Intuitively, if $S_1 \leq S_2$ then a name of type S_1 can safely be used where a name of type S_2 is expected (**safe sustitutability**)

Enhancing Compatibility via Subtyping

We may relate S with T_1 and T_2, using a **subtyping** relation.

- Notation: $S_1 \leq S_2$ (read: S_1 is a subtype of S_2)
- Intuitively, if $S_1 \leq S_2$ then a name of type S_1 can safely be used where a name of type S_2 is expected (**safe sustitutability**)
- Consider the session types (dual to the client types T_1, T_2):

$$S_1 = \ \& \begin{cases} \mathsf{add}: & ?\mathtt{Real}.?\mathtt{Real}.!\mathtt{Real}.\mathsf{end} \\ \mathsf{eq}: & ?\mathtt{Int}.?\mathtt{Int}.!\mathtt{Bool}.\mathsf{end} \end{cases}$$

$$S_2 = \ \& \big\{ \mathsf{add}: \ ?\mathtt{Real}.?\mathtt{Real}.!\mathtt{Real}.\mathsf{end}$$

- We have that:
  $S_1 \leq S$: it is safe to receive integers if reals are supported
  $S_2 \leq S$: it is safe to deal with clients that don't know all options

Subtyping, Formally

For finite session types we may inductively define:

$$\frac{}{\mathsf{end} \leq \mathsf{end}} \qquad \frac{U_1 \leq U_2 \qquad S_1 \leq S_2}{!\,U_2.\,S_1 \ \leq \,!\,U_1.\,S_2} \qquad \frac{U_1 \leq U_2 \qquad S_1 \leq S_2}{?\,U_1.\,S_1 \ \leq \,?\,U_2.\,S_2}$$

$$\frac{I \subseteq J \quad \forall i \in I.\, S_i \leq T_i}{\&\{l_i : S_i\}_{i\in I} \ \leq \ \&\{l_j : T_j\}_{j\in J}} \qquad \frac{J \subseteq I \quad \forall j \in J.\, S_j \leq T_j}{\oplus\{l_j : S_j\}_{j\in J} \ \leq \ \oplus\{l_i : T_i\}_{i\in I}}$$

In our examples:

- $\&\{\mathsf{add} : S_1\} \leq \&\{\mathsf{add} : T_1, \mathsf{eq} : T_2\}$, provided $S_1 \leq T_1$.
- $?\mathtt{Int}.\,?\mathtt{Int}.\,!\mathtt{Bool}.\,\mathsf{end} \leq \,?\mathtt{Real}.\,?\mathtt{Real}.\,!\mathtt{Bool}.\,\mathsf{end}$, provided $\mathtt{Int} \leq \mathtt{Real}$.

Notice

- $\leq$ concerns substitutability of names implementing protocols. Safe substitutability of processes (programs) is also possible.

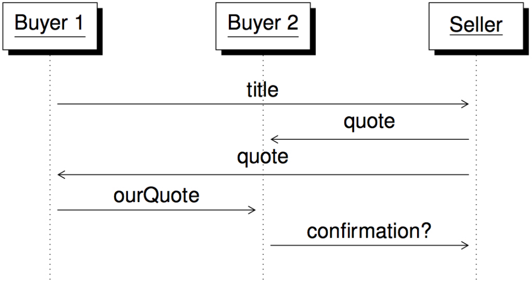
Outline

From Binary to Multiparty Protocols

- Binary session types organize interactions between exactly two partners. Multiple participants follow disjoint protocols.
- In many scenarios, however, three or more partners must interact along the **same session protocol**.
- Decomposing such **multiparty protocols** into binary sessions is not always possible — crucial sequencing information may be lost.

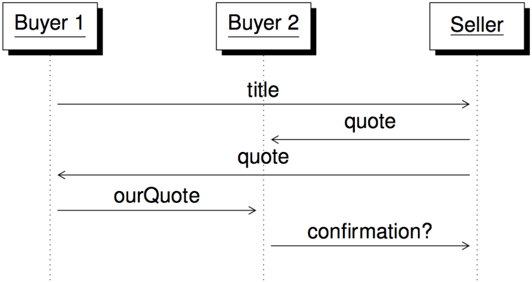
The Need for Sequencing Information

- A two-buyer protocol, similar to the one discussed earlier:

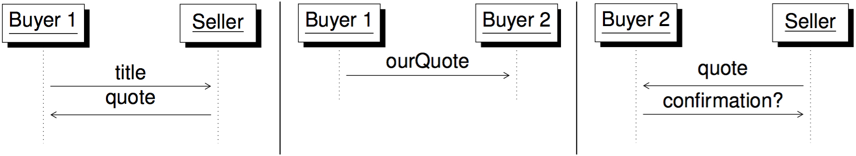


The Need for Sequencing Information

- A two-buyer protocol, similar to the one discussed earlier:

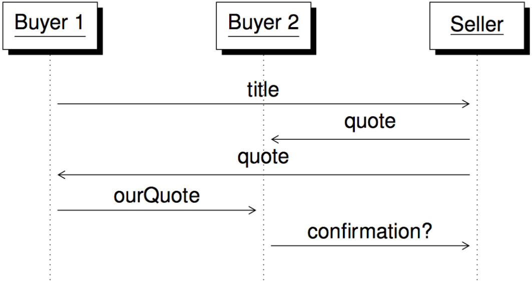


- A decomposition as binary protocols may appear plausible...

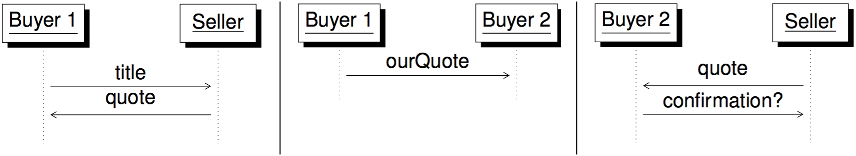


The Need for Sequencing Information

- A two-buyer protocol, similar to the one discussed earlier:



- A decomposition as binary protocols may appear plausible...



- ... but misses key sequencing between unrelated partners.

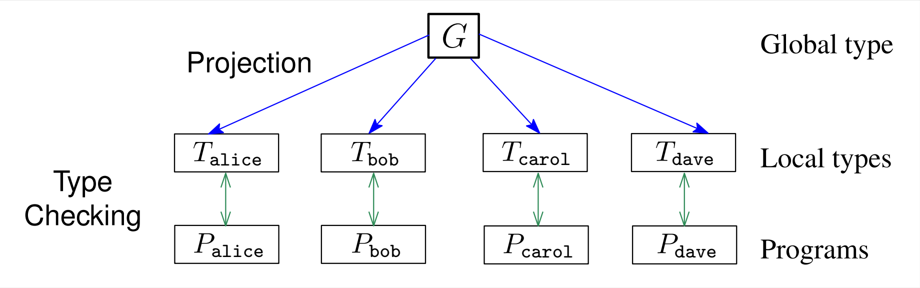
Multiparty Session Types (MPSTs)

A methodology for **decentralized** specification, development, and validation of protocols between multiple participants:

Multiparty Session Types (MPSTs)

A methodology for **decentralized** specification, development, and validation of protocols between multiple participants:

- A **global type**: overall description of the multiparty protocol
- A series of **local types**, one for each participant, obtained from the global type using a **projection function**
- End-point **implementations** can be developed using local types as a reference for (local) validation (e.g. type-checking)



The Syntax of Multiparty Session Types

Let U denote the type for transmittable values.

- Global types:

$$
\begin{array}{rll}
G ::= & \mathrm{p} \to \mathrm{q} : \langle U \rangle . G & \text{Value exchange} \\
\mid & \mathrm{p} \to \mathrm{q} : \{ l_i : G_i \}_{i \in I} & \text{Branching} \\
\mid & \mu t . G \quad \mid \quad t & \text{Recursion} \\
\mid & \text{end} & \text{Terminated global protocol}
\end{array}
$$

- Local types:

$$
\begin{array}{rll}
T ::= & !\langle \mathrm{p}, U \rangle . T & \text{Send value to } \mathrm{p} \\
\mid & ?\langle \mathrm{p}, U \rangle . T & \text{Receive value from } \mathrm{p} \\
\mid & \oplus\langle \mathrm{p}, \{ l_i : T_i \}_{i \in I} \rangle & \text{Select from options offered by } \mathrm{p} \\
\mid & \&\langle \mathrm{p}, \{ l_i : T_i \}_{i \in I} \rangle & \text{Offer labeled options to } \mathrm{p} \\
\mid & \mu t . T \ \mid \ t \ \mid \ \mathtt{end} & \text{Recursion / Terminated Protocol}
\end{array}
$$

Projection

The **projection** of global type G onto participant $\mathtt{r}$, denoted $G{\restriction}\mathtt{r}$:

- $$(\mathtt{p} \to \mathtt{q} : \langle U \rangle . G'){\restriction}\mathtt{r} = \begin{cases} !\langle \mathtt{q}, U \rangle.(G'{\restriction}\mathtt{r}) & \text{if } \mathtt{r} = \mathtt{p} \\ ?\langle \mathtt{p}, U \rangle.(G'{\restriction}\mathtt{r}) & \text{if } \mathtt{r} = \mathtt{q} \\ G'{\restriction}\mathtt{r} & \text{otherwise} \end{cases}$$
- $$(\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i\in I}){\restriction}\mathtt{r} = \begin{cases} \oplus\langle \mathtt{q}, \{l_i : (G_i{\restriction}\mathtt{r})\}_{i\in I} \rangle & \text{if } \mathtt{r} = \mathtt{p} \\ \&\langle \mathtt{p}, \{l_i : (G_i{\restriction}\mathtt{r})\}_{i\in I} \rangle & \text{if } \mathtt{r} = \mathtt{q} \\ G_j{\restriction}\mathtt{r} & \text{if } \mathtt{r} \neq \mathtt{p}, \mathtt{r} \neq \mathtt{q}, j \in I \text{ and} \\ & G_k{\restriction}\mathtt{r} = G_l{\restriction}\mathtt{r}, \text{ for all } k, l \in I \end{cases}$$
- $$(\mu t. G'){\restriction}\mathtt{r} = \begin{cases} \mu t.(G'{\restriction}\mathtt{r}) & \text{if } G'{\restriction}\mathtt{r} \neq t \\ \mathtt{end} & \text{otherwise} \end{cases} \qquad t{\restriction}\mathtt{r} = t$$
- $\text{end}{\restriction}\mathtt{r} = \mathtt{end}$

This is a bit too rigid - why?

The Two-Buyer Protocol, Revisited (1/3)

Alice and Bob cooperate in buying a book from Seller.

1. Alice sends a book title to Seller, who sends a quote back.
2. Alice checks with Bob whether he can contribute in buying the book.
3. Alice uses the answer from Bob to interact with Seller, either
   a) completing the payment and arranging delivery details
   b) canceling the transaction

The Two-Buyer Protocol, Revisited (2/3)

A **single** global protocol G between Alice, Bob, and Seller:

$$
\begin{aligned}
G = \text{Alice} \to \text{Seller} : \;& \langle \texttt{book} \rangle. \\
\text{Seller} \to \text{Alice} : \;& \langle \texttt{quote} \rangle. \\
\text{Alice} \to \text{Bob} : \;& \langle \texttt{cost} \rangle. \\
\text{Bob} \to \text{Alice} : \;& \{\, \text{share} : \quad \text{Alice} \to \text{Bob} : \langle \texttt{ok} \rangle. \\
& \qquad\qquad\quad\; \text{Alice} \to \text{Seller} : \langle \texttt{paym} \rangle.\text{end} \\
& \quad\;\; \text{close} : \quad \text{Alice} \to \text{Bob} : \langle \texttt{bye} \rangle. \\
& \qquad\qquad\quad\; \text{Alice} \to \text{Seller} : \langle \texttt{bye} \rangle.\text{end} \\
& \;\}
\end{aligned}
$$

where $\texttt{book}$, $\texttt{quote}$, $\texttt{cost}$, $\texttt{ok}$, $\texttt{paym}$, $\texttt{bye}$, and $\texttt{close}$ are all base types. Also, for simplicity, we assume that $\texttt{paym} = \texttt{close} = \texttt{str}$.

The Two-Buyer Protocol, Revisited (3/3)

The **projections** of G onto Alice, Bob, and Seller:

$$G\restriction\text{Alice} = \;!\langle\text{Seller}, \texttt{book}\rangle.?\langle\text{Seller}, \texttt{quote}\rangle.!\langle\text{Bob}, \texttt{cost}\rangle.\\ \&\langle\text{Bob}, \{\text{share} : !\langle\text{Bob}, \texttt{ok}\rangle.\\ !\langle\text{Seller}, \texttt{paym}\rangle.\texttt{end}\\ \text{close} : !\langle\text{Bob}, \texttt{bye}\rangle.\texttt{end}\}\,\rangle$$

$$G\restriction\text{Bob} = \;?\langle\text{Alice}, \texttt{cost}\rangle.\\ \oplus\langle\text{Alice}, \{\text{share} : ?\langle\text{Alice}, \texttt{ok}\rangle.\texttt{end}\\ \text{close} : ?\langle\text{Alice}, \texttt{bye}\rangle.\texttt{end}\}\,\rangle$$

$$G\restriction\text{Seller} = \;?\langle\text{Alice}, \texttt{book}\rangle.!\langle\text{Alice}, \texttt{quote}\rangle.\\ ?\langle\text{Alice}, \texttt{paym}/\texttt{close}\rangle.\texttt{end}$$

Taking Stock (1/2)

Binary session types

- Describe protocols between **exactly two partners**
- A session type describes the (possibly infinite) sequence of actions that a given participant performs
- Compatibility defined in terms of session type duality
- Enhancements of compatibility via subtyping

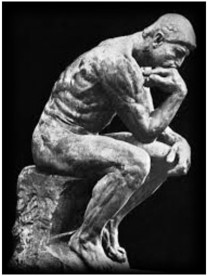
Multiparty session types

- Describe protocols between **more than two partners**
- A global type describes the overall interaction scenario. Local types: binary session types + participant identities.
- Global type projection into local types enforces compatibility. Not all global types are well-formed (i.e., implementable).
- Enhancements via subtyping extend to local types

Outline

Are They Related?

Programmer



Logician

Yes, They Are!

Haskell Curry

William Howard

The **Curry-Howard isomorphism**: an intimate and tight relation between logic and computation:

Propositions	as	Types
Proofs	as	Programs
Simplification of Proofs	as	Program Evaluation

A remarkable correspondence!

Curry-Howard: Significance

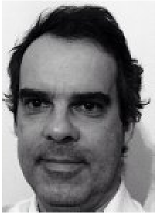
Haskell Curry

William Howard

Viewing “propositions as types, proofs as programs” has important consequences:

- Some aspects of everyday programming are **absolute**
- Understand computation through logic (and vice versa!)

Curry-Howard Today



Luís Caires

Frank Pfenning

- Until recently, the CH isomorphism was limited to sequential programs in the functional paradigm
- In 2010, Luís Caires and Frank Pfenning showed that CH can be extended to concurrent, message-passing programs:

Propositions in **Linear Logic**	as	Session Types
Proofs	as	π-calculus processes
Simplification of Proofs	as	Process Reduction

Linear Logic, Informally (1/3)[1]

- Proposed by Jean-Yves Girard (1987)
- Classical logic deals with stable truths:

  if A and $A \Rightarrow B$ then B, but A still holds

- Example:
  - $A =$ 'Tomorrow is June 22nd'
  - $B =$ 'John will swim'
  - $A \Rightarrow B =$ 'If tomorrow is June 22nd, then John will swim'
- So, if tomorrow is June 22nd, then John will swim.
  This doesn't change the fact that tomorrow will be June 22nd.

[1] Based on slides by Beniamino Accattoli.

Linear Logic, Informally (2/3)

- However, with consumable resources (money, food, etc), classical implications are wrong.
- Example:
  - $A =$ 'John has (only) 5 Euros'
  - $B =$ 'John has a pack of cigarettes'
  - $A \Rightarrow B =$ 'For his 5 Euros, John gets a pack of cigarettes'
- In the classical world, if John buys the cigarettes then he will still have the 5 Euros!

Linear Logic, Informally (3/3)

In Linear Logic:

- Implication consumes hypothesis to produce conclusions
- Linear implications are actions
- Not a new kind of logic, but a refinement of classic logic
- Two conjunctions ($\otimes$ and $\&$), two disjunctions ($⅋$ and $\oplus$), and two modalities for duplicating and discarding resources ($!$ and $?$)
- Connectives are multiplicative ($\otimes$ and $⅋$) and additive ($\&$ and $\oplus$)
- Intuition: multiplicatives denote simultaneous occurrence of resources, whereas additives denote alternative occurrence

Outline

Logic Foundations for Session Types

Linear Logic for Concurrency [Caires&Pfenning'10]

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

A Synchronous π-calculus (2-ary)

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
	$\mid$	$x.\texttt{case}(P, Q)$	branching: offers a choice at x
	$\mid$	$x.\texttt{inl}; P$	select left at x, continue as P
	$\mid$	$x.\texttt{inr}; P$	select right at x, continue as P
	$\mid$	$[x \leftrightarrow y]$	forwarder: fuses 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 (n-ary)

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: fuses 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:

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

Closed under structural congruence, noted $\equiv$.

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

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

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

Session Types as Linear Logic Props

The type syntax 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 Props

The type syntax 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

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

$\Delta \vdash$	$P :: z : \mathbf{1}$	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 sets 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, contraction]

(Names u_i, x_j, z pairwise distinct.)

Example: PDF Conversion Service

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

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

- A process which offers a linear conversion service:

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

- A user which depends on the server:

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

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

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

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 read as well-typed process composition, based on restriction and parallel composition.

Some Typing Rules

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

Some Typing Rules

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

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

Some Typing Rules

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

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

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

Some Typing Rules

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

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

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

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

Some Typing Rules

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

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

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

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

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

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

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

$$\longrightarrow$$

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

Shared Cut as Process Reduction

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

$$\longrightarrow$$

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

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* $\Gamma; \Delta \vdash Q :: z : A$.

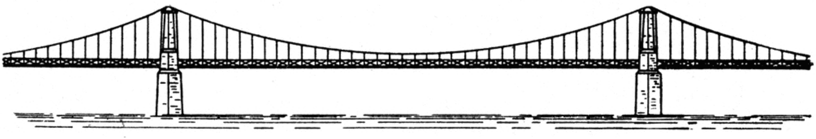
- Process reductions map to principal cut reductions
- Derived properties: communication safety and session fidelity.

For any P, define $\mathit{live}(P)$ 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* $\mathit{live}(P)$ *then exists a* Q *such that* $P \longrightarrow Q$.

Multiparty STs Within Binary STs



First analysis of multiparty sessions within binary session types

- Based on linear logic foundations [Caires&Pfenning'10]
- Relates standard formulations [Honda,Yoshida,Carbone'08]
- Simple and extensible (polymorphism, recursion, asynchrony)

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]

An Open Problem

Can MPSTs Be Reduced Into BSTs?

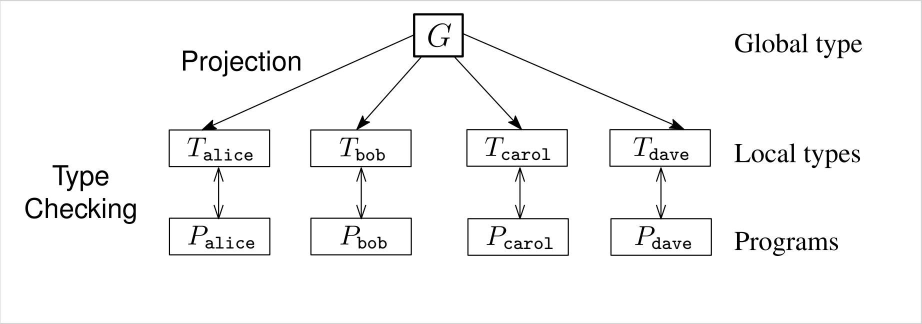
- A reduction would be insightful and practically useful
- Practice suggests MPSTs are more expressive than BSTs
- **Challenge**: Decompose global specs into binary pieces
  - preserving sequencing information
  - avoiding communication errors
  - retaining significance of standard models

A Positive Answer

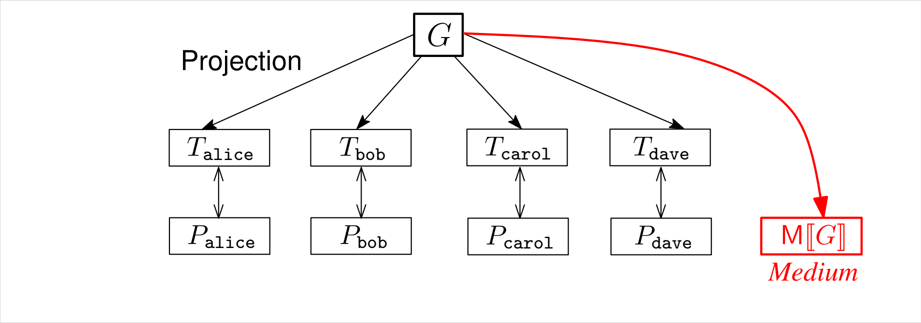
In a recent work (FORTE'16), we have presented a **two-way correspondence** between

- 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

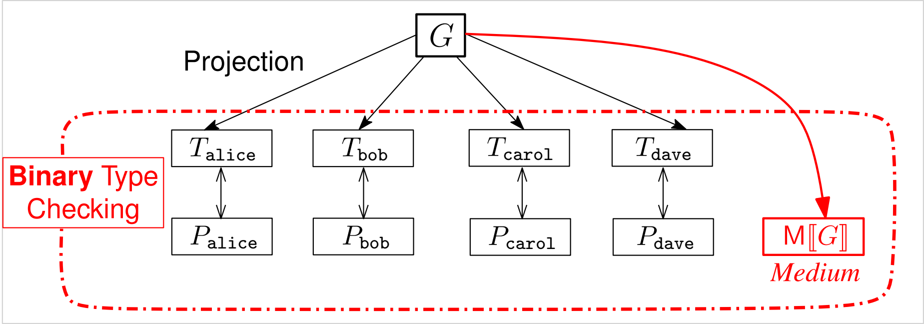


Our Approach: Medium Processes



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

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

- $\mathsf{M}[\![\, \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{\mathtt{l}_i\langle U_i\rangle.G_i\}_{i\in I} \,]\!] =$

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

- $\mathsf{M}[\![\mathsf{end}]\!] = \mathbf{0}$

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

Taking Stock (2/2)

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

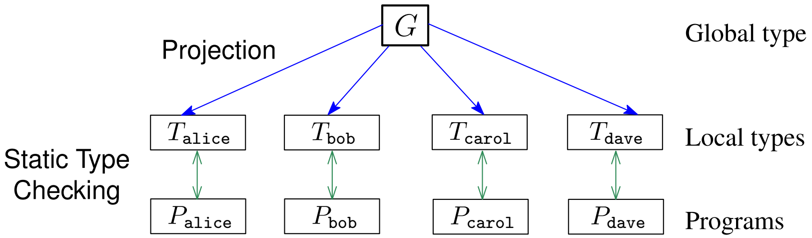
Further Topics

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

- Integration into programming languages (object-oriented, functional, and imperative)
- Connections with **automata theory**
- Synchronous / asynchronous **communication disciplines**
- **Security properties** (secure information flow, access control)
- Different forms of **liveness properties** (progress, deadlock-freedom, and lock-freedom)
- Connections with models of **exceptions**, **reversibility**, **run-time monitoring** and **adaptation**

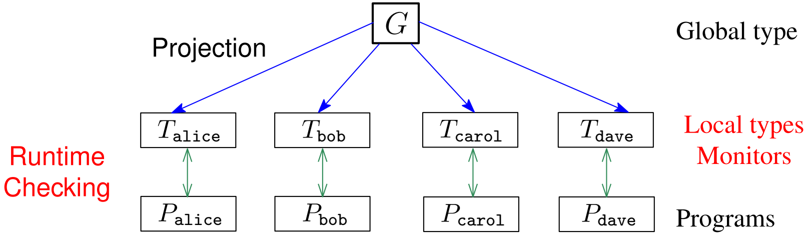
Session Types for Runtime Verification

- The original (and most studied) use of session types is as a **static verification technique** for message-passing programs
- Problem: many components cannot be type-checked.
- Session types can be also used to enforce **runtime verification**.
- Idea: Use each local type as a **monitor** to ensure that the (local) protocol is correctly followed, and to react in case of problems.



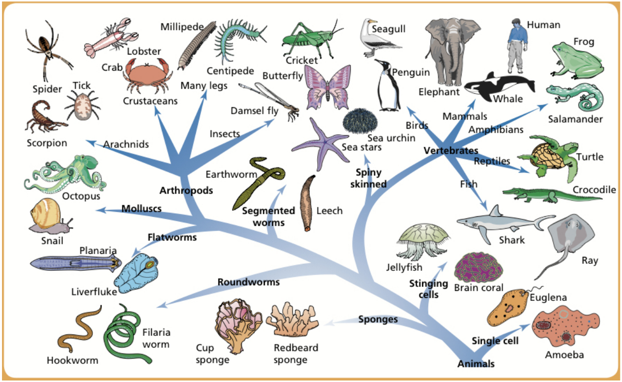
Session Types for Runtime Verification

- The original (and most studied) use of session types is as a **static verification technique** for message-passing programs
- Problem: many components cannot be type-checked.
- Session types can be also used to enforce **runtime verification**.
- Idea: Use each local type as a **monitor** to ensure that the (local) protocol is correctly followed, and to react in case of problems.



(See works by Ancona et al. on dynamic protocol checking for MAS.)

A Pressing Research Challenge



- Many different frameworks of behavioral type systems exist
- Their precision and features vary ostensibly
- There are as many notions of correctness as there are behavioral type systems!

Addressing The Challenge

A recently awarded research grant (NWO VIDI):

- Goal: A unified theory of correctness for message-passing concurrency
- Approach: Use the Curry-Howard correspondence for Concurrency as **objective yardstick** in (formal) comparisons, given as results of **relative expressiveness**
- Initial results promising!
- Impact: Interoperable tools for communicating programs

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.

Type-based Communication Correctness in Multi-agent Systems

Part II: Type Systems for Concurrency and Logical Foundations

Jorge A. Pérez

Bernoulli Institute for Mathematics, Computer Science, and AI
University of Groningen, The Netherlands

www.jperez.nl

20th European Agent Systems Summer School (EASSS 2018)