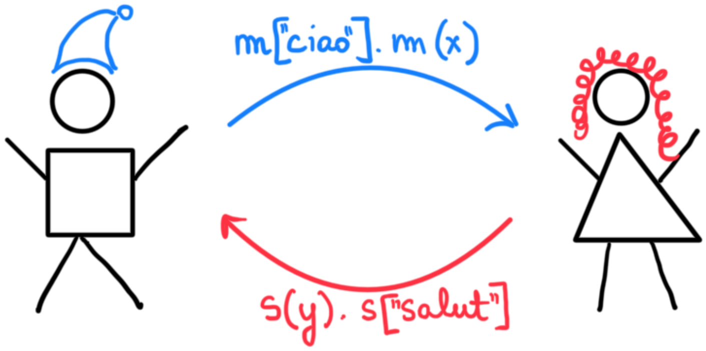
SESSION TYPES

Lecture 1: Basic Concepts

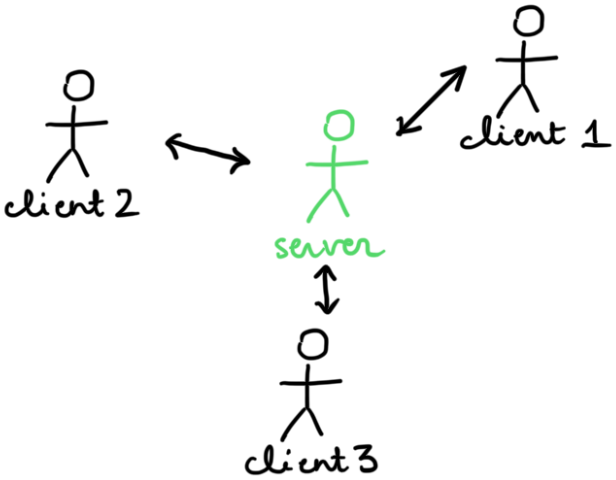
MGS 2024

Matteo Acclavio ⊗ Sonia Marin



Message-passing concurrency:

Processes that compute by exchanging messages along channels.



Session Types are type-theoretic specifications of communication protocols, so that protocol implementations can be verified by compile-time type checking in a programming language.

They were introduced by Kohei Honda and further developed by Takeuchi, Kubo and Vasconcelos.

It has grown into a large, active research area (ST30 workshop).

The theory of session types makes some assumptions about the underlying communication mechanism.

→ point-to-point communication channels
when a message is sent it is received by a single receiver

→ reliable message delivery
no messages are lost

→ preserved order of messages

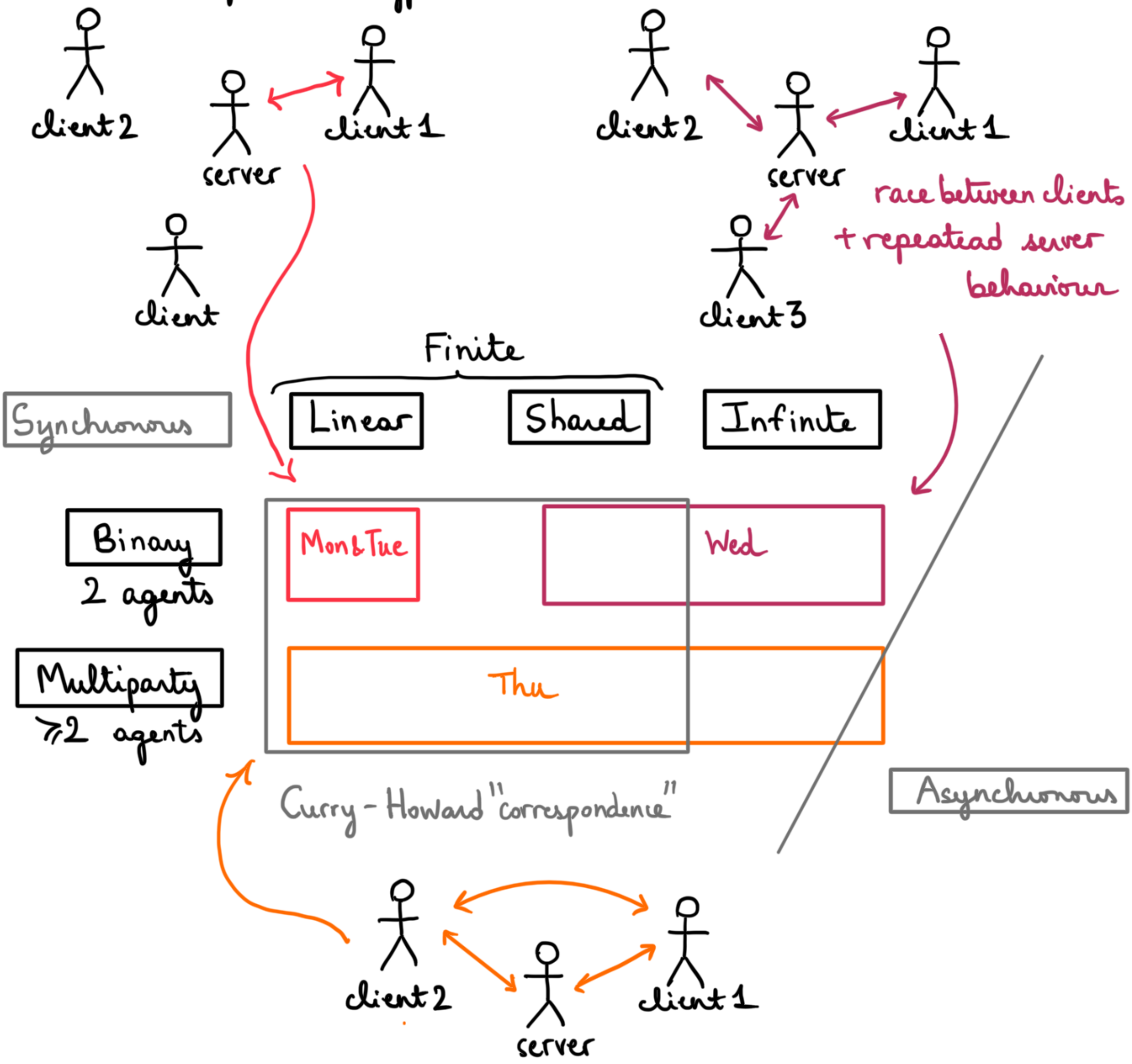
→ synchronous communication
sender and receiver synchronise on every message

Some properties can then be guaranteed by the type system itself:

→ no communication mismatch
on a channel, when the owner of one endpoints sends, the owner of the other endpoint is ready to receive

→ session fidelity
the sequence and types of messages on a channel match its session type

The flavours of session types:



References:

Today's Lecture

- Honda, Vasconcelos & Kubo (ESOP 1998)
  Language primitives and type discipline for structured communication-based programming
- Vasconcelos (Information & Communication 2012)
  Fundamentals of session types

First paper

- Honda (CONCUR 1993)
  Types for dyadic interactions

Further read

- Milner (CUP 1999)
  Communicating and mobile systems : the π-calculus

π-calculus with sessions

for now...

A session is a series of $\boxed{\text{reciprocal}}$ interactions between $\boxed{\text{two parties}}$ and serves as a unit of abstraction for describing interaction.

Each party owns one endpoint of the communication channel.

The goal of session types is to structure communication between concurrent agent.

a variant of

It is built on top of the pi-calculus : a calculus to describe processes computing in parallel and communicating concurrently.

It has operators for sending/receiving messages, parallel execution, scoping of channels...

endpoints ← this means a channel will be of the form uv

Base sets :

	denoted by
channel (variables)	$u, v, \dots$
(expression) variables	$x, y, \dots$
values	$c := u \mid n$
expressions	$e ::= c \mid x \mid e + e$

natural numb.

Syntax of Processes :

threads

$P ::= \text{inact}$ ← $\boxed{\text{terminated}}$ process (inaction)

$\mid u().P$ ← $\boxed{\text{waiting}}$ for channel to close before continuing as process P

$\mid u[\,].P$ ← $\boxed{\text{closing}}$ channel before continuing as process P

$\mid u(x).P$ ← $\boxed{\text{input}}$ process receives, from channel endpoint u, a value that it uses to replace variable x before continuing as process P

x bound in P

$| \ u[e].P$ ← **output** process sends result of evaluating expression e on channel endpoint u before continuing as process P

u and v bound in P

$| \ (\nu uv)\, P$ ← **scope restriction** binds endpoints u and v and establishes them as co-variables

NU

$| \ (P \mid Q)$ ← **parallel composition** allows processes P and Q to proceed concurrently

Co-variables: two endpoints of a communication channel

Idea: some part of the process writes on one
some part reads on the other

Example:

$$\left.\begin{array}{l} P = u[1].u(y).u[\,].P' \\[1em] Q = v(x).v[x+1].v(\,).\mathrm{inact} \end{array}\right\} (\nu uv)\,(P \mid Q)$$

Delegation: channel endpoints can be sent and received as messages

Idea: delegate the processing of a request to another process
which allows for changing the network structure

Example:

$$P = (\nu ab)\,(w[a].y[b].\,(w[\,].\mathrm{inact} \mid y[\,].\mathrm{inact})\,) \qquad w\ x,y,z \in fv$$

$$Q = x(u).u[s].x(\,).u[\,].\mathrm{inact} \qquad R = z(v).z(\,).v(t).v(\,).R'$$

Dynamics

To describe the behaviour and interaction of processes, we define their <u>operational semantics</u>

① Structural congruence :

in order to factor out syntactic differences that are behaviourally irrelevant

defined as the smallest relation that includes :

- $(P \mid Q) \equiv (Q \mid P)$ | commutative
- $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ | associative
- $P \mid \mathbf{inact} \equiv P$ inact neutral for |

- $(\nu uv)(P \mid Q) \equiv (\nu uv) P \mid Q$ scope extrusion
  if $u, v \notin fv(Q)$

- $(\nu uv)\ \mathbf{inact} \equiv \mathbf{inact}$
- $(\nu uv)\ P \equiv (\nu vu) P$
- $(\nu uv)(\nu wz)\ P \equiv (\nu wz)(\nu uv)\ P$

② Operational semantics :

defined as a binary relation → on processes :

- $(\nu uv)(\overbrace{u(x).P}^{\text{waiting to receive on u}} \mid \overbrace{v[e].Q}^{\text{ready to send on v}}) \longrightarrow (\nu uv)(P\overset{\text{substitution}}{[c/x]} \mid Q)$ if $e \downarrow \underline{\underline{c}}$

  (u and v co-variables

  (percists for future communication

- $(\nu uv)(\overbrace{u().P}^{\text{waiting to close}} \mid \overbrace{v[].Q}^{\text{ready to close}}) \longrightarrow (P \mid Q)$

  (to be removed since channel closing

- $\dfrac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}$ — $\dfrac{P \longrightarrow Q}{(\nu uv) P \longrightarrow (\nu uv) Q}$ — $\dfrac{P \equiv P' \quad P \longrightarrow Q \quad Q \equiv Q'}{P' \longrightarrow Q'}$

Example:

$$(\nu pq)\,(p[1].p(y).p[\,].P' \mid q(x).q[x+1].q().\mathrm{inact}) \rightarrow P'[y/2]$$

cf. Exercises

Processes can get stuck in different ways. (errors, deadlocks, ...)

Runtime error: when two threads which are trying to communicate disagree about the "form" of their next step.

Example:

$$(\nu uv)\,(u[\,].\mathrm{inact} \mid v[w].P) \not\rightarrow$$

The aim of the type system is to guarantee that a typable process cannot reduce to an error in any number of steps.

Type system

A session type describes the communication operations that can be performed on one endpoint of a communication channel

① Need to assign types to channel endpoints.

→ Evolution of endpoint types: the type of each endpoint changes through a typing derivation

② Need to control sharing of channel endpoints.

→ Linear type system: a channel endpoint whose type is linear must occur in exactly one thread but may occur many times within that thread

③ Need to guarantee matching communication

→ **Duality**: whenever two endpoints are bound together as co-variables, their types are dual

We begin with types for message passing and terminated processes:

$$T ::= S \mid \mathsf{nat}$$

$S ::= \mathsf{close}$ $\longleftarrow$ Communication has finished, only remaining action is closing the channel.

$\mid \ \mathsf{wait}$ $\longleftarrow$ communication is over, but channel will be closed by other endpoint

$\mid \ [T] \blacktriangleleft S$ $\longleftarrow$ message of type T can be sent on the channel which then must be used according to type S

$\mid \ (T) \blacktriangleleft S$ $\longleftarrow$ message of type T can be received on channel which then must be used according to type S

Duality:
$$(\mathsf{wait})^{\perp} := \mathsf{close} \qquad (\mathsf{close})^{\perp} := \mathsf{wait}$$
$$((T) \blacktriangleleft S) := [T] \blacktriangleleft S^{\perp} \qquad ([T] \blacktriangleleft S) := (T) \blacktriangleleft S^{\perp}$$

Judgement: $\underset{\substack{\uparrow \\ u_1 : S_1, \dots u_n : S_n,\, x_1 : T_1, \dots x_m : T_m}}{\Gamma} \vdash P$ "process P is well-typed under context Γ"

Process typing:

$$\frac{}{\cdot \vdash \mathsf{inact}} \ (\text{INACT})$$

$$\frac{\Gamma \vdash P}{\Gamma, u : \mathsf{wait} \vdash u().P} \ (\text{WAIT}) \qquad \frac{\Gamma \vdash P}{\Gamma, u : \mathsf{close} \vdash u[\,].P} \ (\text{CLOSE})$$

$$\frac{\Gamma,\ y:T,\ u:S \vdash P}{\Gamma,\ u:(T)\blacktriangleleft S \vdash u(y).P}\ (\text{RECV})$$

$$\frac{\Gamma \Vdash e:T \qquad \Gamma,\ u:S \vdash P}{\Gamma,\ u:[T]\blacktriangleleft S \vdash u[e].P}\ (\text{SEND})$$

$$\frac{n \in \mathbb{N}}{\Gamma \Vdash n:\text{nat}}$$

$$\frac{\Gamma \Vdash e:\text{nat} \qquad \Gamma \Vdash e':\text{nat}}{\Gamma \Vdash e+e':\text{nat}}$$

$$\frac{}{\Gamma,\ x:T \Vdash x:T}$$

$$\frac{\Gamma \vdash P \qquad \Delta \vdash Q}{\Gamma,\ \Delta \vdash (P\,|\,Q)}\ (\text{PAR})$$

$$\frac{\Gamma,\ x:S,\ y\ S^{\perp} \vdash P}{\Gamma \vdash (\nu xy)P}\ (\text{RES})$$

Example:

$$p:\underbrace{[\text{nat}]\blacktriangleleft(\text{nat})\blacktriangleleft\text{close}}_{S_p} \vdash \underbrace{p[1].p(y).p[\ ].P'}_{P}$$

$$q:\underbrace{(\text{nat})\blacktriangleleft[\text{nat}]\blacktriangleleft\text{wait}}_{S_q} \vdash \underbrace{q(x).q[x+1].q().\text{inact}}_{Q}$$

$$\cfrac{\cfrac{\begin{array}{c}\vdash P' \\ \vdots \\ p:S_p \vdash P\end{array} \qquad \begin{array}{c}\overline{\vdash \text{inact}} \\ \vdots \\ q:S_q \vdash Q\end{array}}{p:S_p,\ q:S_q \vdash P\,|\,Q}\ (\text{PAR}) \qquad S_p = S_q^{\perp}}{\bullet \vdash (\nu pq)\,(\underbrace{p[1].p(y).p[\ ].P'}_{P}\,|\,\underbrace{q(x).q[x+1].q().\text{inact}}_{Q})}\ (\text{RES})$$