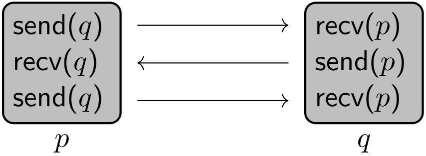
Asynchrony and Choreographies

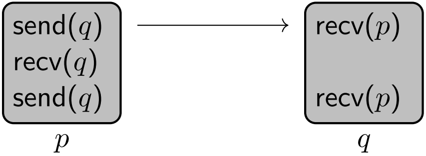
Languages, Systems, and Data Seminar

9 June 2023

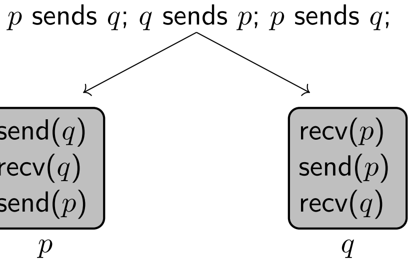
Concurrent and distributed systems



Concurrent and distributed systems



Choreographic programming



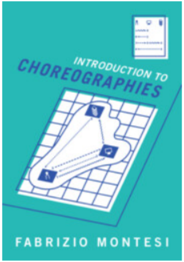
- One centralized program
- Centralized program is compiled via endpoint projection
- Deadlock freedom

Choreographic programming

- Choreographies are the definition of what we want to happen
- Process programs as definitions of how the processes are going to collectively make it happen

Next : A small choreographic language

Introduction to Choreographies by Fabrizio Montesi



Syntax

$$C ::= I; C \mid \theta$$

$$I ::= p.e \to q.x \mid p.x := e$$

$$e ::= v \mid x \mid f(\overrightarrow{e})$$

$\overrightarrow{e}$ ranges over sequences of expressions $e_1, e_2, \ldots, e_n$

Stores and transition labels

- Σ : choreographic store, maps a process name to the process's local store. E.g. $\Sigma(q)$ is q's local store
- $p.v \to q$: p sends value v to q
- $\tau@p$: internal step is performed at p

Process names

$$pn(I; C) \triangleq pn(I) \cup pn(C)$$

$$pn(p.e \to q.x) \triangleq \{p, q\}$$

$$pn(p.x := e) \triangleq \{p\}$$

$$pn(p.v \to q) \triangleq \{p, q\}$$

$$pn(\tau@p) \triangleq \{p\}$$

Semantics

$$\boxed{\langle C, \Sigma\rangle \xrightarrow{\mu} \langle C', \Sigma'\rangle}$$

$$\frac{\Sigma(p) \vdash e \downarrow v}{\langle p.x := e; C, \Sigma\rangle \xrightarrow{\tau @ p} \langle C, \Sigma[p.x \mapsto v]\rangle}\ \text{[Local]}$$

$$\frac{\Sigma(p) \vdash e \downarrow v}{\langle p.e \to q.x; C, \Sigma\rangle \xrightarrow{p.v \to q} \langle C, \Sigma[q.x \mapsto v]\rangle}\ \text{[Com]}$$

$$\frac{\langle C, \Sigma\rangle \xrightarrow{\mu} \langle C', \Sigma'\rangle \qquad pn(I) \cap pn(\mu) = \emptyset}{\langle I; C, \Sigma\rangle \xrightarrow{\mu} \langle I; C', \Sigma'\rangle}\ \text{[Delay]}$$

Example (choreography)

$$\boxed{C \triangleq p.v_1 \to r_1.z \ ; \ p.v_2 \to r_2.y \ ; \ p.v_3 \to r_3.z}$$

$$\begin{array}{ll}
\langle p.v_1 \to r_1.x \ ; \ p.v_2 \to r_2.y \ ; \ p.v_3 \to r_3.z, \Sigma\rangle & [\mathsf{Com}] \\
\quad \downarrow_{p.v_1 \to r_1} & \\
\langle p.v_2 \to r_2.y \ ; \ p.v_3 \to r_3.z, \Sigma[r_1.x \mapsto v_1]\rangle & [\mathsf{Com}] \\
\quad \downarrow_{p.v_2 \to r_2} & \\
\langle p.v_3 \to r_3.z, \Sigma[r_1.x \mapsto v_1, r_2.y \mapsto v_2]\rangle & [\mathsf{Com}] \\
\quad \downarrow_{p.v_3 \to r_3} & \\
\langle \theta, \Sigma[r_1.x \mapsto v_1, r_2.y \mapsto v_2, r_3.z \mapsto v_3]\rangle &
\end{array}$$

Process programs

- Implementing the choreography requires translating it to appropriate process programs (via end-point-projection)

Syntax for processes

$$P ::= I; P \mid \theta$$

$$I ::= p!e \mid p?x \mid x := e$$

$$e ::= v \mid x \mid f(\overrightarrow{e})$$

End-point-projection

$$[\![p.e \to q.x; C]\!]_r \triangleq \begin{cases} q!e; [\![C]\!]_r \text{ if } r = p \\ p?x; [\![C]\!]_r \text{ if } r = q \\ [\![C]\!]_r \text{ otherwise} \end{cases}$$

$$[\![p.x := e; C]\!]_r \triangleq \begin{cases} x := e; [\![C]\!]_r \text{ if } r = p \\ [\![C]\!]_r \text{ otherwise} \end{cases}$$

$$N \triangleq p_1[P_1] | p_2[P_2] | \ldots | p_n[P_n]$$

Example

$$C \triangleq p.v_1 \to r_1.z; p.v_2 \to r_2.y; p.v_3 \to r_3.z$$

$$N \triangleq p[r_1!v_1; r_2!v_2; r_3!v_3] \mid r_1[p?x] \mid r_3[p?z] \mid r_2[p?y]$$

Semantics of processes

$$\boxed{\langle N,\Sigma\rangle \xrightarrow{\mu} \langle N',\Sigma'\rangle}$$

$$\frac{\Sigma(p)\vdash e\downarrow v}{\langle p[x{:=}e;P],\Sigma\rangle \xrightarrow{\tau @ p} \langle p[P],\Sigma[p.x\mapsto v]\rangle}\text{[Local]}$$

$$\frac{\Sigma(p)\vdash e\downarrow v}{\langle p[q!e;P] \,|\, q[p?x;Q],\Sigma\rangle \xrightarrow{p.v\to q} \langle p[P] \,|\, q[Q],\Sigma[q.x\mapsto v]\rangle}\text{[Com]}$$

$$\frac{\langle N,\Sigma\rangle \xrightarrow{\mu} \langle N',\Sigma'\rangle}{\langle N|M,\Sigma\rangle \xrightarrow{\mu} \langle N'|M,\Sigma'\rangle}\text{[Par]}$$

Example (processes)

$$
\begin{array}{rl}
\langle p[r_1!v_1; r_2!v_2; r_3!v_3] \mid r_1[p?x] \mid r_3[p?z] \mid r_2[p?y], \Sigma\rangle & \text{[Com]} \\
\downarrow_{p.v_1 \to r_1} & \\
\langle p[r_2!v_2; r_3!v_3] \mid r_1[\theta] \mid r_3[p?z] \mid r_2[p?y], \Sigma'\rangle & \text{[Com]} \\
\downarrow_{p.v_2 \to r_2} & \\
\langle p[r_3!v_3] \mid r_1[\theta] \mid r_3[p?z] \mid r_2[\theta], \Sigma''\rangle & \text{[Com]} \\
\downarrow_{p.v_3 \to r_3} & \\
\langle p[\theta] \mid r_1[\theta] \mid r_3[\theta] \mid r_2[\theta], \Sigma'''\rangle & \equiv \langle \theta, \Sigma'''\rangle
\end{array}
$$

$$\boxed{p[\theta] = \theta \qquad p[P] | \theta = p[P] \qquad \theta | \theta = \theta}$$

Synchronous communication

- So far our choreographic language is synchronous
- A sending action blocks the sender until it can interact with a compatible receiving action at the intended receiver

Asynchronous communication

- Allow a sending action to be executed without waiting for the receiver to be ready

Message state and queue

$$K[q \mapsto \overrightarrow{m}](p) \triangleq \begin{cases} \overrightarrow{m} \text{ if } p = q \\ K(p) \text{ otherwise} \end{cases} \qquad \text{where } \overrightarrow{m} = m_1, m_2 \ldots$$

- Each message m_i is of form (p, v)
- The message from the sender can be immediately stored in a message queue of the receiver
- The intended receiver can later retrieve the message

Syntax

$$C ::= I; C$$

$$I ::= p.e \to q.x \mid p.x := e \mid \boldsymbol{p \rightsquigarrow q.x}$$

$$e ::= v \mid x \mid f(\overrightarrow{e})$$

Transition labels

- ~~$p.v \to q$: p sends value v to q~~
- $\tau@p$: internal step is performed at p
- $p.v \to q!$: denotes p has sent value v to q
- $p.v \to q?$: denotes q received value v from p

Process names

$$pn(I;C) \triangleq pn(I) \cup pn(C)$$
$$pn(p.e \to q.x) \triangleq \{p, q\}$$
$$pn(p.x := e) \triangleq \{p\}$$
$$pn(p \rightsquigarrow q.x) \triangleq \{q\}$$

$$\cancel{pn(p.v \to q) \triangleq \{p, q\}}$$
$$pn(p.v \to q!) \triangleq \{p\}$$
$$pn(p.v \to q?) \triangleq \{q\}$$
$$pn(\tau @ p) \triangleq \{p\}$$

Semantics

$$\boxed{\langle C,\Sigma,K\rangle \xrightarrow{\mu} \langle C',\Sigma',K'\rangle}$$

$$\frac{\Sigma(p)\vdash e\downarrow v}{\langle p.x{:=}e;C,\Sigma,K\rangle \xrightarrow{\tau@p} \langle C,\Sigma[p.x\mapsto v],K\rangle}\;[\text{Local}]$$

$$\frac{\Sigma(p)\vdash e\downarrow v \qquad K(q)=\overrightarrow{m}}{\langle p.e\to q.x;C,\Sigma\rangle \xrightarrow{p.v\to q!} \langle p\rightsquigarrow q.x;C,\Sigma,K[q.x\mapsto \overrightarrow{m},(p,v)]\rangle}\;[\text{Send-Val}]$$

$$\frac{K(q)=(p,v),\overrightarrow{m}}{\langle p\rightsquigarrow q.x;C,\Sigma,K\rangle \xrightarrow{p.v\to q?} \langle C,\Sigma[q.x\mapsto v],K[q\mapsto \overrightarrow{m}]\rangle}\;[\text{Recv-Val}]$$

$$\frac{\langle C,\Sigma,K\rangle \xrightarrow{\mu} \langle C',\Sigma',K'\rangle \qquad pn(I)\cap pn(\mu)=\emptyset}{\langle I;C,\Sigma,K\rangle \xrightarrow{\mu} \langle I;C',\Sigma',K'\rangle}\;[\text{Delay}]$$

Example(choreography)

$$\langle p.v_1 \rightarrow r_1.x; p.v_2 \rightarrow r_2.y; p.v_3 \rightarrow r_3.z, \Sigma, K\rangle$$
$$\downarrow_{p.v_1 \rightarrow r_1!} \qquad \text{[Send-Val]}$$
$$\langle p \rightsquigarrow r_1.x; p.v_2 \rightarrow r_2.y; p.v_3 \rightarrow r_3.z, \Sigma, K[r_1 \mapsto (p, v_1)]\rangle$$
$$\downarrow_{p.v_2 \rightarrow r_2!} \qquad \text{[Send-Val]}$$
$$\langle p \rightsquigarrow r_1.x; p \rightsquigarrow r_2.y; p.v_3 \rightarrow r_3.z, \Sigma, K[r_1 \mapsto (p, v_1), r_2 \mapsto (p, v_2)]\rangle$$
$$\downarrow_{p.v_2 \rightarrow r_2?} \qquad \text{[Recv-Val]}$$
$$\langle p \rightsquigarrow r_1.x; p.v_3 \rightarrow r_3.z, \Sigma[r_2.y \mapsto v_2], K[r_1 \mapsto (p, v_1)]\rangle$$
$$\downarrow_{p.v_3 \rightarrow r_3!} \qquad \text{[Send-Val]}$$
$$\langle p \rightsquigarrow r_1.x; p \rightsquigarrow r_3.z, \Sigma[r_2.y \mapsto v_2], K[r_1 \mapsto (p, v_1), r_3 \mapsto (p, v_3)]\rangle$$
$$\downarrow_{p.v_3 \rightarrow r_3?} \qquad \text{[Recv-Val]}$$
$$\langle p \rightsquigarrow r_1.x, \Sigma[r_2.y \mapsto v_2, r_3.z \mapsto v_3], K[r_1 \mapsto (p, v_1)]\rangle$$
$$\downarrow_{p.v_1 \rightarrow r_1?} \qquad \text{[Recv-Val]}$$
$$\langle \theta, \Sigma[r_2.y \mapsto v_2, r_3.z \mapsto v_3, r_1.x \mapsto v_1], K\rangle$$

Syntax for processes

$$P ::= I; P \mid \theta$$

$$I ::= p!e \mid p?x \mid x := e$$

$$e ::= v \mid x \mid f(\overrightarrow{e})$$

$$N \triangleq p_1[P_1] | \mathrm{p_2[P_2]} | \ldots \mid \mathrm{p_n[P_n]}$$

End-point-projection

$$[\![p.e \to q.x; C]\!]_r \triangleq \begin{cases} q!e; [\![C]\!]_r \text{ if } r = p \\ p?x; [\![C]\!]_r \text{ if } r = q \\ [\![C]\!]_r \text{ otherwise} \end{cases}$$

$$[\![p.x := e; C]\!]_r \triangleq \begin{cases} x := e; [\![C]\!]_r \text{ if } r = p \\ [\![C]\!]_r \text{ otherwise} \end{cases}$$

$$[\![p \rightsquigarrow q.x; C]\!]_r \triangleq \begin{cases} p?x; [\![C]\!]_r \text{ if } r = q \\ [\![C]\!]_r \text{ otherwise} \end{cases}$$

Semantics of processes

$$\boxed{\langle N,\Sigma,K\rangle \xrightarrow{\mu} \langle N',\Sigma',K'\rangle}$$

$$\frac{\Sigma(p)\vdash e\downarrow v}{\langle p[x{:=}e;P],\Sigma,K\rangle \xrightarrow{\tau @ p} \langle p[P],\Sigma[p.x\mapsto v],K\rangle}\text{[Local]}$$

$$\frac{\Sigma(p)\vdash e\downarrow v \qquad K(q)=\overrightarrow{m}}{\langle p[q!e;P],\Sigma,K\rangle \xrightarrow{p.v\to q!} \langle p[P],\Sigma,K[q\mapsto \overrightarrow{m},(p,v)]\rangle}\text{[Send-Val]}$$

$$\frac{K(q)=(p,v),\overrightarrow{m}}{\langle q[p?x;Q],\Sigma,K\rangle \xrightarrow{p.v\to q?} \langle q[Q],\Sigma[q.x\mapsto v],K[q\mapsto \overrightarrow{m}]\rangle}\text{[Recv-Val]}$$

$$\frac{\langle N,\Sigma,K\rangle \xrightarrow{\mu} \langle N',\Sigma',K'\rangle}{\langle N|M,\Sigma,K\rangle \xrightarrow{\mu} \langle N'|M,\Sigma',K'\rangle}\text{[Par]}$$

Example

$$C \triangleq p.v_1 \to r_1.z; p.v_2 \to r_2.y; p.v_3 \to r_3.z$$

$$N \triangleq p[r_1!v_1; r_2!v_2; r_3!v_3] \mid r_1[p?x] \mid r_3[p?z] \mid r_2[p?y]$$

Example (processes)

$$\langle p[r_1!v_1; r_2!v_2; r_3!v_3] \mid r_1[p?x] \mid r_3[p?z] \mid r_2[p?y], \Sigma, K\rangle$$

$$\downarrow_{p.v_1 \to r_1!} \qquad \text{[Send-Val]}$$

$$\langle p[r_2!v_2; r_3!v_3] \mid r_1[p?x] \mid r_3[p?z] \mid r_2[p?y], \Sigma, K[r_1 \mapsto (p, v_1)]\rangle$$

$$\downarrow_{p.v_2 \to r_2!} \qquad \text{[Send-Val]}$$

$$\langle p[r_3!v_3] \mid r_1[p?x] \mid r_3[p?z] \mid r_2[p?y], \Sigma, K[r_1 \mapsto (p, v_1), r_2 \mapsto (p, v_2)]\rangle$$

$$\downarrow_{p.v_2 \to r_2?} \qquad \text{[Recv-Val]}$$

$$\langle p[r_2!v_3] \mid r_1[p?x] \mid r_3[p?z] \mid r_2[\theta], \Sigma[r_2.y \mapsto v_2], K[r_1 \mapsto (p, v_1)]\rangle$$

$$\downarrow_{p.v_3 \to r_3!} \qquad \text{[Send-Val]}$$

$$\langle p[\theta] \mid r_1[p?x] \mid r_3[p?z], \Sigma[r_2.y \mapsto v_2], K[r_1 \mapsto (p, v_1), r_3 \mapsto (p, v_3)]\rangle$$

Receiving order is maintained

$$\langle p.v_1 \to r_1.x; p.v_2 \to r_1.y, \Sigma, K\rangle$$

$$\downarrow_{p.v_1 \to r_1!}$$

$$\downarrow_{p.v_2 \to r_1!}$$

$$\langle p \rightsquigarrow r_1.x; p \rightsquigarrow r_1.y, \Sigma, K'\rangle$$

$$\not\downarrow_{p.v_2 \to r_1?}$$

$$\boxed{pn(p \rightsquigarrow r_1.x) \cap pn(p.v_2 \to r_1?) \neq \emptyset}$$

Receiving order is maintained

$$\langle p_1.v_1 \to r_1.x; p_2.v_2 \to r_1.y, \Sigma, K\rangle$$

$$\downarrow_{p_1.v_1 \to r_1!}$$

$$\downarrow_{p_2.v_2 \to r_1!}$$

$$\langle p_1 \rightsquigarrow r_1.x; p_2 \rightsquigarrow r_1.y, \Sigma, K'\rangle$$

$$\not\downarrow_{p_2.v_2 \to r_1?}$$

$$\boxed{pn(p_1 \rightsquigarrow r_1.x) \cap pn(p_2.v_2 \to r_1?) \neq \emptyset}$$

Deadlock freedom

Consider the choreography $p \rightsquigarrow q.x$, and let's assume the messaging state does not have a message from p for q — hence q will get stuck

This can never happen as $p \rightsquigarrow q.x$ is generated via [Send-Val] rule from some term $p.v \to q.x$