Encoding Asynchrony in Choreographies

Luís Cruz-Filipe and Fabrizio Montesi

Department of Mathematics and Computer Science, University of Southern Denmark

Choreographic Programming

is a paradigm for developing concurrent programs that are deadlock-free by construction, by first programming communications declaratively, and then synthesising process implementations automatically.

$$\begin{tikzcd} C \arrow[r, dash, "\text{execution}"] \arrow[d, dash, "\text{projection}"'] & C' \arrow[d, dash, "\text{projection}"] \\ P \arrow[r, dash, "\text{execution}"'] & P' \end{tikzcd}$$

Choreographies
(abstract, easy to read, deadlock free)

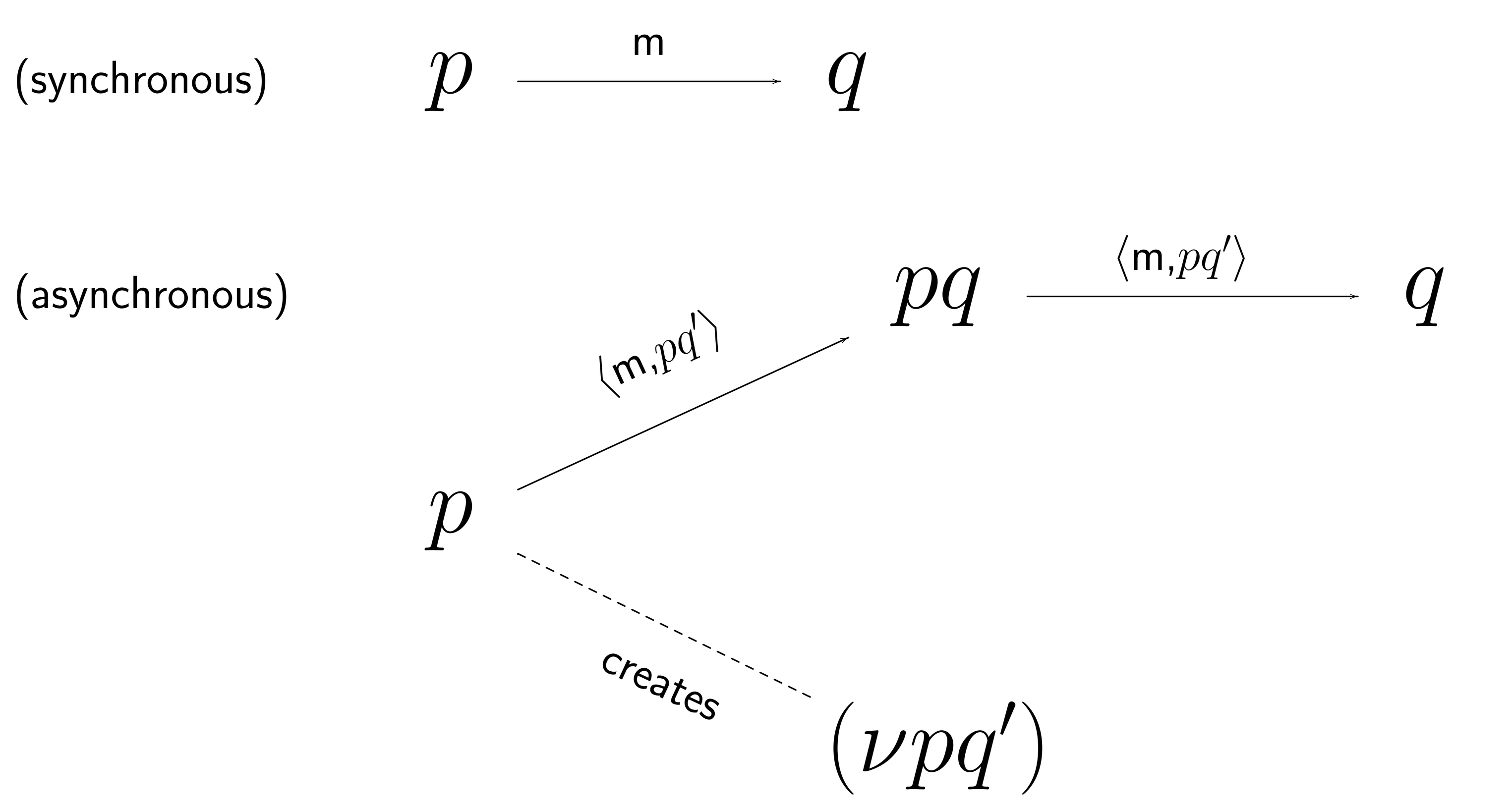
Implementations
(concrete, hard to parse, undecidable properties)

Communications

Communications in choreographies are typically synchronous. Yet choreographies are widely used both for the specification and the programming of concurrent and distributed software architectures, which use asynchronous communications.

In order to model asynchrony in choreographies, several authors have proposed relying on additional technical machinery, such as ad-hoc syntactic terms, alternative semantics, or sophisticated behavioural equivalences.
We show that such extensions are not needed for choreography languages that support primitives for process spawning and name mobility.
Instead, we can encode asynchronous communications in choreographies themselves, yielding a simpler approach.

The main idea



Initially, p and q agree on an initial communication process pq. Before sending a message, p creates a new process pq' that will store the next message. Then p sends the message and the name pq' to pq, which stores this information until q is ready to receive it. In the meantime, p can continue executing – and even send the next message to q through the new auxiliary process.

An example

We define a choreography where a buyer, Alice (a), purchases a product from a seller (s) through her bank (b).

Two-buyer protocol	1. $a.title$ `->` s; 2. $s.price$ `->` a; 3. $s.price$ `->` b; 4. if $b.ok$ then $s.book$ `->` a; else $\mathbf{0}$

Asynchronous version	setup. a start as^0; s start sa^0; s start sb^0; $a : as^0$ `<->` s; $s : sa^0$ `<->` a; $s : sb^0$ `<->` b; 1. $a.title$ `->` as^0; a start as^1; $a : as^1$ `<->` as^0; $as^0.as^1$ `->` s; $as^0.s$ `->` as^1; $as^0.title$ `->` s; 2. $s.price$ `->` sa^0; s start sa^1; $s : sa^1$ `<->` sa^0; $sa^0.sa^1$ `->` a; $sa^0.a$ `->` sa^1; $sa^0.price$ `->` a; 3. $s.price$ `->` sb^0; s start sb^1; $s : sb^1$ `<->` sb^0; $sb^0.sb^1$ `->` b; $sb^0.b$ `->` sb^1; $sb^0.price$ `->` b; 4. if $b.ok$ then $s.book$ `->` a; else $\mathbf{0}$

We applied our construction to make communications 1–3 asynchronous.

The formal definition

Let C be a choreography over a set of processes $\mathcal{P}$. The encoding uses a parameter $M : \mathcal{P} \times \mathcal{P} \to \mathbb{N}$, which is a function keeping track of the auxiliary communication channels.
We start with M_0 such that $M_0(p,q) = 0$ for all p and q, and setup all initial auxiliary channels.

$$\{\!\!\{C\}\!\!\} = \left\{p\,\mathsf{start}\,pq^0;\ p : q\,\texttt{<->}pq^0\right\}_{p \neq q};\ \{\!\!\{C\}\!\!\}_{M_0}$$

The term $p\,\mathsf{start}\,pq^0$ creates a fresh process pq^0, only known to its creator (in this case p). In $p : q\,\texttt{<->}pq^0$, p communicates the name of q to pq^0 and conversely, so that these processes are now able to communicate directly.
The rest of the encoding is structural, changing only communication actions as suggested in the picture on the left. For simplicity, we write pq^M for $pq^{M(p,q)}$ and pq^{M+} for $pq^{M(p,q)+1}$.

$$\begin{aligned}\{\!\!\{p.e\ \texttt{->}\ q; C\}\!\!\}_M = {} & p\,\mathsf{start}\,pq^{M+};\ p.e\ \texttt{->}\ pq^M;\ p : pq^M\ \texttt{<->}\ pq^{M+}; \\ & pq^M.q\texttt{->}pq^{M+};\ pq^M.pq^{M+}\texttt{->}q;\ pq^M.e\ \texttt{->}\ q; \\ & \{\!\!\{C\}\!\!\}_{M[(p,q)\mapsto M(p,q)+1]}\end{aligned}$$

Due to the out-of-order execution allowed by the choreography semantics, p can proceed after the actions in the first line are completed.

Semantics

We use a transition semantics over triples G, C, σ, where C is a choreography, G is the graph of connections, describing which pairs of processes are allowed to communicate, and σ is a state, describing which values are stored at each process.

Theorem

Choreographies in this model are deadlock-free.

Theorem

Let $p \in \mathsf{pn}(C)$ and $pq \in \mathsf{pn}(\{\!\!\{C\}\!\!\}) \setminus \mathsf{pn}(C)$. If $G, \{\!\!\{C\}\!\!\}, \sigma \to^* G', C_1, \sigma_1 \to G', C_2, \sigma_2$ where in the last transition a value v is sent from p to pq, then there exist G'', C_3, σ_3, C_4 and σ_4 such that $G', C_2, \sigma_2 \to^* G'', C_3, \sigma_3 \to G'', C_4, \sigma_4$ and in the last transition the same value v is sent from pq to some process $q \in \mathsf{pn}(C)$.

Theorem

If $G, \{\!\!\{C\}\!\!\}_M, \sigma \to^* G_1, C_1, \sigma_1$, then there exist C', σ' and σ'' such that $G, C, \sigma \to^* G, C', \sigma'$, and $G_1, C_1, \sigma_1 \to^* G', \{\!\!\{C'\}\!\!\}_M, \sigma''$, and σ' and σ'' coincide on the values stored at $\mathsf{pn}(C)$.

LaTeX TikZposter