# *a turing-complete choreography calculus*

luís cruz-filipe

(joint work with fabrizio montesi)

department of mathematics and computer science
university of southern denmark

abcd meeting, university of glasgow
september 9th, 2016

# *outline*

*the zoo of communication*

*communication & computation*

*practical consequences*

# *models of communicating systems*

*process calculi*

$\pi$-calculus and its variants

- low-level modeling of communication
- too technical for many purposes
- many interesting fragments are undecidable

# *models of communicating systems*

*process calculi*

$\pi$-calculus and its variants

- low-level modeling of communication
- too technical for many purposes
- many interesting fragments are undecidable

*choreographies*

- global view of the system
- directed communication (from alice to bob)
- deadlock-free by design
- compilable to process calculi

# *choreographies and computation*

$\rightsquigarrow$ trivially turing-complete
(arbitrary computation at each process)

# *choreographies and computation*

- trivially turing-complete
(arbitrary computation at each process)
- typically geared towards applications
(many complex primitives)

## *choreographies and computation*

$\rightsquigarrow$ trivially turing-complete
(arbitrary computation at each process)

$\rightsquigarrow$ typically geared towards applications
(many complex primitives)

*focus* communication

- reduce local computation to a minimum
- reduce system primitives to a minimum
- how far can we go?

## *choreographies and computation*

$\rightsquigarrow$ trivially turing-complete
(arbitrary computation at each process)

$\rightsquigarrow$ typically geared towards applications
(many complex primitives)

*focus* communication

- reduce local computation to a minimum
- reduce system primitives to a minimum
- how far can we go?

*$\pi$-calculus* direct encoding of $\lambda$-calculus is unsatisfactory

- counter-intuitive notion of computation
- data and programs at the same level

# *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)

## *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)

$$\mathsf{M} \overset{\text{computes}}{\rightsquigarrow} \mathsf{M}'$$

## *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)

$$
\begin{tikzcd}
\mathsf{M} \arrow[rr, rightsquigarrow, "\text{computes}"] \arrow[d, Rightarrow, "\text{compiles}"'] & & \mathsf{M}' \arrow[d, Rightarrow, "\text{compiles}"] \\
\mathsf{P}_{\mathsf{M}} & & \mathsf{P}_{\mathsf{M}'}
\end{tikzcd}
$$

# *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)

$$
\begin{tikzcd}
\mathsf{M} \arrow[r, squiggly, "\text{computes}"] \arrow[d, Rightarrow, "\text{compiles}"'] & \mathsf{M}' \arrow[d, Rightarrow, "\text{compiles}"] \\
\mathsf{P}_{\mathsf{M}} \arrow[r, squiggly, "\text{computes}"] & \mathsf{P}_{\mathsf{M}'}
\end{tikzcd}
$$

## *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)

$$
\begin{tikzcd}
 & \mathsf{C} & & \mathsf{C}' \\
\mathsf{M} \arrow[ur, dotted, "\text{embedded}"] \arrow[rr, squiggly, "\text{computes}"] & & \mathsf{M}' \arrow[ur, dotted, "\text{embedded}"'] &
\end{tikzcd}
$$

# *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)

$$
\begin{tikzcd}
 & \mathsf{C} \arrow[rr, squiggly, "\text{computes}"] & & \mathsf{C}' \\
\mathsf{M} \arrow[ur, dotted, "\text{embedded}"] \arrow[rr, squiggly, "\text{computes}"] & & \mathsf{M}' \arrow[ur, dotted, "\text{embedded}"'] &
\end{tikzcd}
$$

# *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)

$$
\begin{tikzcd}
\mathsf{C} \arrow[rrr, rightsquigarrow, "\text{computes}"] \arrow[dd, Rightarrow, "\text{compiles}"'] & & & \mathsf{C}' \arrow[dd, Rightarrow, "\text{compiles}"] \\
 & & & \\
\mathsf{P}_{\mathsf{C}} & & & \mathsf{P}_{\mathsf{C}'}
\end{tikzcd}
$$

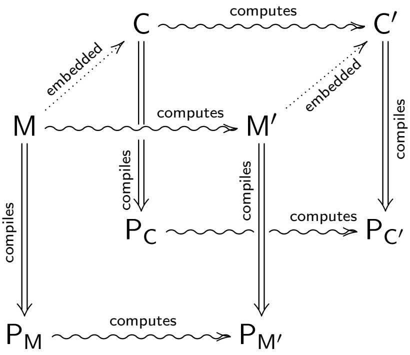
# *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)

$$\begin{tikzcd} \mathsf{C} \arrow[rr, squiggly, "\text{computes}"] \arrow[dd, Rightarrow, "\text{compiles}"'] & & \mathsf{C}' \arrow[dd, Rightarrow, "\text{compiles}"] \\ & & \\ \mathsf{P}_{\mathsf{C}} \arrow[rr, squiggly, "\text{computes}"] & & \mathsf{P}_{\mathsf{C}'} \end{tikzcd}$$

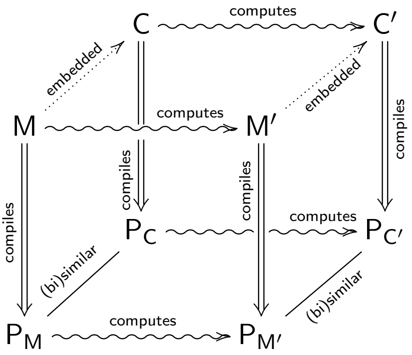
## *our contribution*

- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)



# *our contribution*

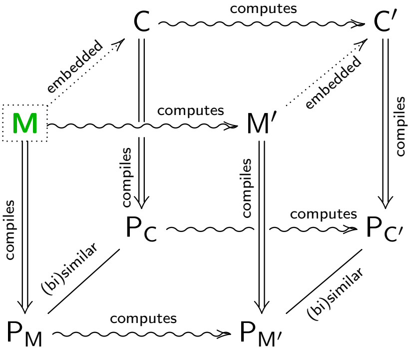
- i/o-based notion of function implementation
- computation by message-passing
- reminiscent of memory models (e.g. urm)



# *our contribution*

focus of this talk:

- minimal choreographies
- their turing completeness



# *outline*

# *typical primitives in choreographies*

- termination
- message passing
- label selection
- conditionals
- recursion
- process creation
- channel creation
- channel passing
- role assignment
- ...

# *typical primitives in choreographies*

- termination
- message passing
- label selection
- conditionals
- recursion
- process creation
- channel creation
- channel passing
- role assignment
- ...

# *typical primitives in choreographies*

- termination
- message passing
- label selection
- conditionals
- recursion
- process creation
- channel creation
- channel passing
- role assignment
- ...

# *minimal choreographies*



$$M ::= \quad \mathbf{0} \mid \eta; M \mid \mathsf{if}\,(\mathsf{p}.* = \mathsf{q}.*)\,\mathsf{then}\, M_1\, \mathsf{else}\, M_2$$
$$\mid \mathsf{def}\, X = M_2\, \mathsf{in}\, M_1 \mid X$$

$$\eta ::= \mathsf{p}.e \to \mathsf{q} \mid \mathsf{p} \to \mathsf{q}[l]$$

$$l ::= \mathrm{L} \mid \mathrm{R}$$

$$e ::= \varepsilon \mid * \mid \mathsf{s} \cdot *$$

# minimal choreographies

*minimal choreographies*

$$M ::= \quad \mathbf{0} \mid \eta; M \mid \mathsf{if}\ (\mathsf{p}.* = \mathsf{q}.*)\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2$$
$$\mid \mathsf{def}\ X = M_2\ \mathsf{in}\ M_1 \mid X$$

$$\eta ::= \mathsf{p}.e \to \mathsf{q} \mid \mathsf{p} \to \mathsf{q}[l]$$

$$l ::= \mathrm{L} \mid \mathrm{R}$$

$$e ::= \varepsilon \mid * \mid \mathsf{s} \cdot *$$

*urm machine* classical model of computation

- similar to physical memory
- memory cells store natural numbers
- memory operations: zero, successor, copy
- jump-on-equal

# minimal choreographies

*minimal choreographies*

$$M ::= \mathbf{0} \mid \eta; M \mid \mathsf{if}\ (\mathsf{p}.* = \mathsf{q}.*)\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2$$
$$\mid \mathsf{def}\ X = M_2\ \mathsf{in}\ M_1 \mid X$$

$$\eta ::= \mathsf{p}.e \to \mathsf{q} \mid \mathsf{p} \to \mathsf{q}[l]$$

$$l ::= \mathrm{L} \mid \mathrm{R}$$

$$e ::= \varepsilon \mid * \mid \mathsf{s} \cdot *$$

*urm machine* classical model of computation

- similar to physical memory
- memory cells store natural numbers $\rightsquigarrow$ processes
- memory operations: zero, successor, copy
- jump-on-equal $\rightsquigarrow$ conditional / loop

# *minimal choreographies*

*minimal choreographies*

$$M ::= \mathbf{0} \mid \eta; M \mid \mathsf{if}\ (\mathsf{p}.* = \mathsf{q}.*)\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2$$
$$\mid \mathsf{def}\ X = M_2\ \mathsf{in}\ M_1 \mid X$$

$$\eta ::= \mathsf{p}.e \to \mathsf{q} \mid \mathsf{p} \to \mathsf{q}[l]$$

$$l ::= \mathrm{L} \mid \mathrm{R}$$

$$e ::= \varepsilon \mid * \mid \mathsf{s} \cdot *$$

*but. . . !* very different computation model

- no centralized control
- no self-change

# minimal choreographies

*minimal choreographies*

$$M ::= \mathbf{0} \mid \eta; M \mid \text{if } (\mathsf{p}.* = \mathsf{q}.*) \text{ then } M_1 \text{ else } M_2$$
$$\mid \text{def } X = M_2 \text{ in } M_1 \mid X$$

$$\eta ::= \mathsf{p}.e \to \mathsf{q} \mid \mathsf{p} \to \mathsf{q}[l]$$

$$l ::= \mathrm{L} \mid \mathrm{R}$$

$$e ::= \varepsilon \mid * \mid \mathsf{s} \cdot *$$

*on selections*

- not needed for computational completeness
- essential (?) for projectability (e.g. to $\pi$-calculus)
- known algorithms for inferring selections

## *implementation of functions*

*state* a *state* of an minimal choreography is a mapping from the set of process names to the set of values

# *implementation of functions*

*state* a *state* of an minimal choreography is a mapping from the set of process names to the set of values

*implementation* choreography $M$ implements $f : \mathbb{N}^n \to \mathbb{N}$ with inputs $p_1, \ldots, p_n$ and output $q$ if:
for every $\sigma$ such that $\sigma(p_i) = \ulcorner x_i \urcorner$,

- if $f(\tilde{x})$ is defined, then $M, \sigma \to^* \mathbf{0}, \sigma'$ and $\sigma'(q) = \ulcorner f(\tilde{x}) \urcorner$
- if $f(\tilde{x})$ is not defined, then $M, \sigma \not\to^* \mathbf{0}$ (diverges)

## *an example: addition*

*addition*
*from* p, q *to* r
*using* t

$$
\begin{array}{l}
\mathsf{def}\ X = \\
\quad \mathsf{if}\ (\mathsf{r}.* = \mathsf{q}.*)\ \mathsf{then} \\
\qquad \mathsf{p}.* \to \mathsf{r};\ \mathbf{0} \\
\quad \mathsf{else} \\
\qquad \mathsf{p}.* \to \mathsf{t};\ \mathsf{t}.(\mathsf{s}\cdot *) \to \mathsf{p}; \\
\qquad \mathsf{r}.* \to \mathsf{t};\ \mathsf{t}.(\mathsf{s}\cdot *) \to \mathsf{r};\ X \\
\mathsf{in}\ \mathsf{t}.\varepsilon \to \mathsf{r};\ X
\end{array}
$$

## *an example: addition*

*addition*
*from* p, q *to* r
*using* t

$$
\begin{array}{l}
\text{def } X = \\
\quad \text{if } (\mathsf{r}.* = \mathsf{q}.*) \text{ then} \\
\quad\quad \mathsf{p}.* \to \mathsf{r};\ \mathbf{0} \\
\quad \text{else} \\
\quad\quad \mathsf{p}.* \to \mathsf{t};\ \mathsf{t}.(\mathsf{s}\cdot *) \to \mathsf{p}; \\
\quad\quad \mathsf{r}.* \to \mathsf{t};\ \mathsf{t}.(\mathsf{s}\cdot *) \to \mathsf{r};\ X \\
\text{in } \mathsf{t}.\varepsilon \to \mathsf{r};\ X
\end{array}
$$

$\leadsto$ does not compile!

- projection of p does not know whether to send a message to r or t
- projection of t does not know whether to wait for a message or terminate

## *an example: addition*

*addition*
*from* p, q *to* r
*using* t

$$
\begin{array}{l}
\mathsf{def}\ X = \\
\qquad \mathsf{if}\ (\mathsf{r}.* = \mathsf{q}.*)\ \mathsf{then}\ \mathsf{r} \to \mathsf{p}[\mathrm{L}];\ \mathsf{r} \to \mathsf{q}[\mathrm{L}];\ \mathsf{r} \to \mathsf{t}[\mathrm{L}]; \\
\qquad\qquad \mathsf{p}.* \to \mathsf{r};\ \mathbf{0} \\
\qquad \mathsf{else}\ \mathsf{r} \to \mathsf{p}[\mathrm{R}];\ \mathsf{r} \to \mathsf{q}[\mathrm{R}];\ \mathsf{r} \to \mathsf{t}[\mathrm{R}]; \\
\qquad\qquad \mathsf{p}.* \to \mathsf{t};\ \mathsf{t}.(\mathsf{s} \cdot *) \to \mathsf{p}; \\
\qquad\qquad \mathsf{r}.* \to \mathsf{t};\ \mathsf{t}.(\mathsf{s} \cdot *) \to \mathsf{r};\ X \\
\mathsf{in}\ \mathsf{t}.\varepsilon \to \mathsf{r};\ X
\end{array}
$$

$\rightsquigarrow$ does not compile!

- projection of p does not know whether to send a message to r or t
- projection of t does not know whether to wait for a message or terminate

# *an example: addition*

*addition*
*from* p, q *to* r
*using* t

$$
\begin{array}{l}
\mathsf{def}\ X = \\
\qquad \mathsf{if}\ (\mathsf{r}.* = \mathsf{q}.*)\ \mathsf{then}\ \mathsf{r} \to \mathsf{p}[\mathrm{L}];\ \mathsf{r} \to \mathsf{q}[\mathrm{L}];\ \mathsf{r} \to \mathsf{t}[\mathrm{L}]; \\
\qquad\qquad \mathsf{p}.* \to \mathsf{r};\ \mathbf{0} \\
\qquad \mathsf{else}\ \mathsf{r} \to \mathsf{p}[\mathrm{R}];\ \mathsf{r} \to \mathsf{q}[\mathrm{R}];\ \mathsf{r} \to \mathsf{t}[\mathrm{R}]; \\
\qquad\qquad \mathsf{p}.* \to \mathsf{t};\ \mathsf{t}.(\mathsf{s} \cdot *) \to \mathsf{p}; \\
\qquad\qquad \mathsf{r}.* \to \mathsf{t};\ \mathsf{t}.(\mathsf{s} \cdot *) \to \mathsf{r};\ X \\
\mathsf{in}\ \mathsf{t}.\varepsilon \to \mathsf{r};\ X
\end{array}
$$

$\rightsquigarrow$ compiles!

- projections of p and t wait for notification from r
- projection of q also needs to be notified

*successor* $S : \mathbb{N} \to \mathbb{N}$ such that $S(x) = x + 1$ for all $x$

# *partial recursive functions i/vi*

*successor* $S : \mathbb{N} \to \mathbb{N}$ such that $S(x) = x + 1$ for all $x$

*implementation*

$$[\![S]\!]^{\mathsf{p} \mapsto \mathsf{q}} = \mathsf{p}.(\mathsf{s} \cdot *) \to \mathsf{q}$$

# *partial recursive functions i/vi*

*successor* $S : \mathbb{N} \to \mathbb{N}$ such that $S(x) = x + 1$ for all $x$

*implementation*

$$[\![S]\!]^{\mathsf{p}\mapsto\mathsf{q}} = \mathsf{p}.(\mathsf{s} \cdot *) \to \mathsf{q}$$

*soundness*

$$\mathsf{p}.(\mathsf{s} \cdot *) \to \mathsf{q}, \{\mathsf{p} \mapsto \ulcorner x \urcorner\} \longrightarrow \mathbf{0}, \begin{Bmatrix} \mathsf{p} \mapsto \ulcorner x \urcorner \\ \mathsf{q} \mapsto \ulcorner x + 1 \urcorner \end{Bmatrix}$$

*zero* $Z : \mathbb{N} \to \mathbb{N}$ such that $S(x) = 0$ for all $x$

*implementation*

$$[\![Z]\!]^{\mathsf{p} \mapsto \mathsf{q}} = \mathsf{p}.\varepsilon \to \mathsf{q}$$

*soundness*

$$\mathsf{p}.\varepsilon \to \mathsf{q}, \{\mathsf{p} \mapsto \ulcorner x \urcorner\} \longrightarrow \mathbf{0}, \begin{Bmatrix} \mathsf{p} \mapsto \ulcorner x \urcorner \\ \mathsf{q} \mapsto \ulcorner 0 \urcorner \end{Bmatrix}$$

# *partial recursive functions iii/vi*

*projections* $P^n_m : \mathbb{N} \to \mathbb{N}$ such that $P^n_m(x_1, \ldots, x_n) = x_m$ for all $\tilde{x}$

*implementation*

$$[\![P^n_m]\!]^{\mathsf{p_1,\ldots,p_n \mapsto q}} = \mathsf{p_m}.* \to \mathsf{q}$$

*soundness*

$$\mathsf{p_m}.* \to \mathsf{q}, \{\mathsf{p_i} \mapsto \ulcorner x_i \urcorner\} \longrightarrow \mathbf{0}, \left\{\begin{matrix} \mathsf{p_i} \mapsto \ulcorner x_i \urcorner \\ \mathsf{q} \mapsto \ulcorner x_m \urcorner \end{matrix}\right\}$$

# *intermezzo: properties of the encoding*

$\rightsquigarrow$ properties we use in inductive constructions

- execution preserves contents of input processes
- all choreographies have exactly one exit point (occurrence of $\mathbf{0}$)

# *intermezzo: properties of the encoding*

$\rightsquigarrow$ properties we use in inductive constructions

- execution preserves contents of input processes
- all choreographies have exactly one exit point (occurrence of $\mathbf{0}$)

*sequential composition* for processes with only one exit point $M \fatsemi M'$ is obtained by replacing $\mathbf{0}$ (in $M$) by $M'$

# *intermezzo: properties of the encoding*

$\rightsquigarrow$ properties we use in inductive constructions

- execution preserves contents of input processes
- all choreographies have exactly one exit point (occurrence of $\mathbf{0}$)

*sequential composition* for processes with only one exit point $M \fatsemi M'$ is obtained by replacing $\mathbf{0}$ (in $M$) by $M'$

$\rightsquigarrow$ works as expected

- if $M, \sigma \rightarrow^* \mathbf{0}, \sigma'$ and $M', \sigma' \rightarrow^* \mathbf{0}, \sigma''$, then $M \fatsemi M', \sigma \rightarrow^* \mathbf{0}, \sigma''$
- if $M, \sigma \rightarrow^* \mathbf{0}, \sigma'$ and $M', \sigma'$ diverges, then $M \fatsemi M', \sigma$ diverges
- if $M, \sigma$ diverges, then $M \fatsemi M', \sigma \rightarrow \mathbf{0}, \sigma''$ diverges

# *partial recursive functions iv/vi*

*composition*

$$g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N} \qquad C(f, \tilde{g}) : \mathbb{N}^n \to \mathbb{N}$$

$$f : \mathbb{N}^k \to \mathbb{N} \qquad \tilde{x} \mapsto f(g_1(\tilde{x}), \ldots, g_k(\tilde{x}))$$

# *partial recursive functions iv/vi*

*composition*

$$g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N} \qquad C(f, \tilde{g}) : \mathbb{N}^n \to \mathbb{N}$$

$$f : \mathbb{N}^k \to \mathbb{N} \qquad \tilde{x} \mapsto f(g_1(\tilde{x}), \ldots, g_k(\tilde{x}))$$

*implementation*

$$[\![C(f, \tilde{g})]\!]_{\ell}^{\mathsf{p}_1, \ldots, \mathsf{p}_n \mapsto \mathsf{q}} = [\![g_1]\!]_{\ell_1}^{\mathsf{p}_1, \ldots, \mathsf{p}_n \mapsto \mathsf{r}'_1} \fatsemi \ldots \fatsemi [\![g_k]\!]_{\ell_k}^{\mathsf{p}_1, \ldots, \mathsf{p}_n \mapsto \mathsf{r}'_k} \fatsemi [\![f]\!]_{\ell_{k+1}}^{\mathsf{r}'_1, \ldots, \mathsf{r}'_k \mapsto \mathsf{q}}$$

# *partial recursive functions iv/vi*

*composition*

$$g_1, \dots, g_k : \mathbb{N}^n \to \mathbb{N} \qquad C(f, \tilde{g}) : \mathbb{N}^n \to \mathbb{N}$$
$$f : \mathbb{N}^k \to \mathbb{N} \qquad \tilde{x} \mapsto f(g_1(\tilde{x}), \dots, g_k(\tilde{x}))$$

*implementation*

$$[\![C(f, \tilde{g})]\!]_{\ell}^{\mathsf{p}_1,\dots,\mathsf{p}_n \mapsto \mathsf{q}} = [\![g_1]\!]_{\ell_1}^{\mathsf{p}_1,\dots,\mathsf{p}_n \mapsto \mathsf{r}'_1} \fatsemi \dots \fatsemi [\![g_k]\!]_{\ell_k}^{\mathsf{p}_1,\dots,\mathsf{p}_n \mapsto \mathsf{r}'_k} \fatsemi [\![f]\!]_{\ell_{k+1}}^{\mathsf{r}'_1,\dots,\mathsf{r}'_k \mapsto \mathsf{q}}$$

⇝ $\mathsf{r}'_\mathsf{i}$ are auxiliary processes numbered from $\ell$: $\mathsf{r}'_\mathsf{i} = \mathsf{r}_{\ell+\mathsf{i}-1}$
in recursive calls we increment the counter:
$\ell_{i+1} = \ell_i + \pi(g_i)$

# partial recursive functions iv/vi

*composition*

$$g_1, \dots, g_k : \mathbb{N}^n \to \mathbb{N} \qquad C(f, \tilde{g}) : \mathbb{N}^n \to \mathbb{N}$$
$$f : \mathbb{N}^k \to \mathbb{N} \qquad \tilde{x} \mapsto f(g_1(\tilde{x}), \dots, g_k(\tilde{x}))$$

*implementation*

$$\llbracket C(f, \tilde{g}) \rrbracket_{\ell}^{\mathsf{p}_1, \dots, \mathsf{p}_n \mapsto \mathsf{q}} = \llbracket g_1 \rrbracket_{\ell_1}^{\mathsf{p}_1, \dots, \mathsf{p}_n \mapsto \mathsf{r}'_1} \fatsemi \dots \fatsemi \llbracket g_k \rrbracket_{\ell_k}^{\mathsf{p}_1, \dots, \mathsf{p}_n \mapsto \mathsf{r}'_k} \fatsemi \llbracket f \rrbracket_{\ell_{k+1}}^{\mathsf{r}'_1, \dots, \mathsf{r}'_k \mapsto \mathsf{q}}$$

*soundness*

$$\llbracket C(f, \tilde{g}) \rrbracket_{\ell}^{\mathsf{p}_1, \dots, \mathsf{p}_n \mapsto \mathsf{q}}, \{\mathsf{p}_\mathsf{i} \mapsto \ulcorner x_i \urcorner\}$$
$$\longrightarrow^* \llbracket f \rrbracket_{\ell_{k+1}}^{\mathsf{r}'_1, \dots, \mathsf{r}'_k \mapsto \mathsf{q}}, \begin{Bmatrix} \mathsf{p}_\mathsf{i} \mapsto \ulcorner x_i \urcorner \\ \mathsf{r}'_\mathsf{j} \mapsto \ulcorner g_j(\tilde{x}) \urcorner \end{Bmatrix}$$

# partial recursive functions iv/vi

*composition*

$$g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N} \qquad C(f, \tilde{g}) : \mathbb{N}^n \to \mathbb{N}$$
$$f : \mathbb{N}^k \to \mathbb{N} \qquad \tilde{x} \mapsto f(g_1(\tilde{x}), \ldots, g_k(\tilde{x}))$$

*implementation*

$$\llbracket C(f, \tilde{g}) \rrbracket_{\ell}^{\mathsf{p}_1, \ldots, \mathsf{p}_n \mapsto \mathsf{q}} = \llbracket g_1 \rrbracket_{\ell_1}^{\mathsf{p}_1, \ldots, \mathsf{p}_n \mapsto \mathsf{r}'_1} \fatsemi \ldots \fatsemi \llbracket g_k \rrbracket_{\ell_k}^{\mathsf{p}_1, \ldots, \mathsf{p}_n \mapsto \mathsf{r}'_k} \fatsemi \llbracket f \rrbracket_{\ell_{k+1}}^{\mathsf{r}'_1, \ldots, \mathsf{r}'_k \mapsto \mathsf{q}}$$

*soundness*

$$\llbracket C(f, \tilde{g}) \rrbracket_{\ell}^{\mathsf{p}_1, \ldots, \mathsf{p}_n \mapsto \mathsf{q}}, \{\mathsf{p}_\mathsf{i} \mapsto \ulcorner x_i \urcorner\}$$
$$\longrightarrow^* \llbracket f \rrbracket_{\ell_{k+1}}^{\mathsf{r}'_1, \ldots, \mathsf{r}'_k \mapsto \mathsf{q}}, \begin{Bmatrix} \mathsf{p}_\mathsf{i} \mapsto \ulcorner x_i \urcorner \\ \mathsf{r}'_\mathsf{j} \mapsto \ulcorner g_j(\tilde{x}) \urcorner \end{Bmatrix}$$
$$\longrightarrow^* \mathbf{0}, \begin{Bmatrix} \mathsf{p}_\mathsf{i} \mapsto \ulcorner x_i \urcorner \\ \mathsf{r}'_\mathsf{j} \mapsto \ulcorner g_j(\tilde{x}) \urcorner \\ \mathsf{q} \mapsto \ulcorner f(\widetilde{g(\tilde{x})}) \urcorner \end{Bmatrix}$$

## partial recursive functions iv/vi

*composition*

$$g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N} \qquad C(f, \tilde{g}) : \mathbb{N}^n \to \mathbb{N}$$
$$f : \mathbb{N}^k \to \mathbb{N} \qquad \tilde{x} \mapsto f(g_1(\tilde{x}), \ldots, g_k(\tilde{x}))$$

*implementation*

$$[\![C(f,\tilde{g})]\!]_{\ell}^{\mathsf{p}_1,\ldots,\mathsf{p}_n \mapsto \mathsf{q}} = [\![g_1]\!]_{\ell_1}^{\mathsf{p}_1,\ldots,\mathsf{p}_n \mapsto \mathsf{r}'_1} \fatsemi \ldots \fatsemi [\![g_k]\!]_{\ell_k}^{\mathsf{p}_1,\ldots,\mathsf{p}_n \mapsto \mathsf{r}'_k} \fatsemi [\![f]\!]_{\ell_{k+1}}^{\mathsf{r}'_1,\ldots,\mathsf{r}'_k \mapsto \mathsf{q}}$$

*soundness*

if $g_j(\tilde{x})$ is undefined the corresponding step diverges and likewise for $f(\widetilde{g(\tilde{x})})$

recursion

$$f : \mathbb{N}^n \to \mathbb{N} \qquad g : \mathbb{N}^{n+2} \to \mathbb{N}$$

$$h = R(f, g) : \mathbb{N}^{n+1} \to \mathbb{N}$$

$$\tilde{x} \mapsto \begin{cases} f(\vec{x}) & x_0 = 0 \\ g(k, h(k, \tilde{x}), \tilde{x}) & x_0 = k + 1 \end{cases}$$

# *partial recursive functions v/vi*

*recursion*

$$f : \mathbb{N}^n \to \mathbb{N} \qquad g : \mathbb{N}^{n+2} \to \mathbb{N}$$

$$h = R(f, g) : \mathbb{N}^{n+1} \to \mathbb{N}$$

$$\tilde{x} \mapsto \begin{cases} f(\vec{x}) & x_0 = 0 \\ g(k, h(k, \tilde{x}), \tilde{x}) & x_0 = k + 1 \end{cases}$$

*implementation*

$$[\![h]\!]^{\mathsf{p}_0,\ldots,\mathsf{p}_n \mapsto \mathsf{q}} =$$

$$\text{def } T = \text{if } \mathsf{r_c}.* = \mathsf{p}_0.* \text{ then } \mathsf{q}'.* \to \mathsf{q};\ \mathbf{0}$$

$$\text{else } [\![g]\!]_{\ell_g}^{\mathsf{r_c},\mathsf{q}',\mathsf{p}_1,\ldots,\mathsf{p}_n \mapsto \mathsf{r_t}} \fatsemi \mathsf{r_t}.* \to \mathsf{q}';$$

$$\mathsf{r_c}.* \to \mathsf{r_t};\ \mathsf{r_t}.(\mathsf{s} \cdot *) \to \mathsf{r_c};\ T$$

$$\text{in } [\![f]\!]_{\ell_f}^{\mathsf{p}_1,\ldots,\mathsf{p}_n \mapsto \mathsf{q}'} \fatsemi \mathsf{r_t}.\varepsilon \to \mathsf{r_c};\ T$$

*recursion*

$$f : \mathbb{N}^n \to \mathbb{N} \qquad g : \mathbb{N}^{n+2} \to \mathbb{N}$$

$$h = R(f, g) : \mathbb{N}^{n+1} \to \mathbb{N}$$

$$\tilde{x} \mapsto \begin{cases} f(\vec{x}) & x_0 = 0 \\ g(k, h(k, \tilde{x}), \tilde{x}) & x_0 = k + 1 \end{cases}$$

*implementation*

$$\llbracket h \rrbracket^{\mathsf{p}_0,\ldots,\mathsf{p}_n \mapsto \mathsf{q}} =$$

$$\begin{aligned} &\mathsf{def}\ T = \mathsf{if}\ \mathsf{r_c}.* = \mathsf{p_0}.*\ \mathsf{then}\ \mathsf{q}'.* \to \mathsf{q};\ \mathbf{0} \\ &\qquad\qquad \mathsf{else}\ \llbracket g \rrbracket_{\ell_g}^{\mathsf{r_c},\mathsf{q}',\mathsf{p}_1,\ldots,\mathsf{p}_n \mapsto \mathsf{r_t}} \fatsemi \mathsf{r_t}.* \to \mathsf{q}'; \\ &\qquad\qquad\qquad \mathsf{r_c}.* \to \mathsf{r_t};\ \mathsf{r_t}.(\mathsf{s} \cdot *) \to \mathsf{r_c};\ T \\ &\mathsf{in}\ \llbracket f \rrbracket_{\ell_f}^{\mathsf{p}_1,\ldots,\mathsf{p}_n \mapsto \mathsf{q}'} \fatsemi \mathsf{r_t}.\varepsilon \to \mathsf{r_c};\ T \end{aligned}$$

*soundness*

by induction (simple)

*minimization*

$$f : \mathbb{N}^{n+1} \to \mathbb{N} \qquad M(f) : \mathbb{N}^n \to \mathbb{N}$$

$$\tilde{x} \mapsto \mu y . f(\vec{x}, y) = 0$$

## *partial recursive functions vi/vi*

*minimization*

$$f : \mathbb{N}^{n+1} \to \mathbb{N} \qquad M(f) : \mathbb{N}^n \to \mathbb{N}$$

$$\tilde{x} \mapsto \mu y . f(\vec{x}, y) = 0$$

*implementation*

$$[\![M(f)]\!]^{\mathsf{p}_1,\ldots,\mathsf{p}_n \mapsto \mathsf{q}} =$$

$$\begin{aligned}
&\mathsf{def}\ T = [\![f]\!]_{\ell_f}^{\mathsf{p}_1,\ldots,\mathsf{p}_n,\mathsf{r_c} \mapsto \mathsf{q}'} \fatsemi \mathsf{r_c}.\varepsilon \to \mathsf{r_z};\\
&\qquad \mathsf{if}\ \mathsf{r_z}.* = \mathsf{q}'.*\ \mathsf{then}\ \mathsf{r_c}.* \to \mathsf{q};\ \mathbf{0}\\
&\qquad \mathsf{else}\ \mathsf{r_c}.* \to \mathsf{r_z};\ \mathsf{r_z}.(\mathsf{s} \cdot *) \to \mathsf{r_c};\ T\\
&\mathsf{in}\ \mathsf{r_z}.\varepsilon \to \mathsf{r_c};\ T
\end{aligned}$$

## *partial recursive functions vi/vi*

*minimization*

$$f : \mathbb{N}^{n+1} \to \mathbb{N} \qquad M(f) : \mathbb{N}^n \to \mathbb{N}$$
$$\tilde{x} \mapsto \mu y. f(\vec{x}, y) = 0$$

*implementation*

$$[\![M(f)]\!]^{\mathsf{p}_1,\dots,\mathsf{p}_n \mapsto \mathsf{q}} =$$
$$\mathrm{def}\ T = [\![f]\!]_{\ell_f}^{\mathsf{p}_1,\dots,\mathsf{p}_n,\mathsf{r_c} \mapsto \mathsf{q}'} \fatsemi \mathsf{r_c}.\varepsilon \to \mathsf{r_z};$$
$$\mathrm{if}\ \mathsf{r_z}.* = \mathsf{q}'.* \ \mathrm{then}\ \mathsf{r_c}.* \to \mathsf{q};\ \mathbf{0}$$
$$\mathrm{else}\ \mathsf{r_c}.* \to \mathsf{r_z};\ \mathsf{r_z}.(\mathsf{s} \cdot *) \to \mathsf{r_c};\ T$$
$$\mathrm{in}\ \mathsf{r_z}.\varepsilon \to \mathsf{r_c};\ T$$

*soundness*

by induction (simple)

## *minimality*

*minimal choreographies*

$$
\begin{aligned}
M ::= \quad & \mathbf{0} \mid \eta; M \mid \mathsf{if}\ (\mathsf{p}.* = \mathsf{q}.*)\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 \\
& \mid \mathsf{def}\ X = M_2\ \mathsf{in}\ M_1 \mid X
\end{aligned}
$$

$$\eta ::= \mathsf{p}.e \to \mathsf{q} \mid \mathsf{p} \to \mathsf{q}[l]$$

$$l ::= \mathrm{L} \mid \mathrm{R}$$

$$e ::= \varepsilon \mid * \mid \mathsf{s} \cdot *$$

- no exit points $\rightsquigarrow$ nothing terminates
- no communication $\rightsquigarrow$ no output
- less expressions $\rightsquigarrow$ cannot compute base cases
- no conditions $\rightsquigarrow$ termination is decidable
- no recursion $\rightsquigarrow$ everything terminates

# *minimality*

*minimal choreographies*

$$M ::= \mathbf{0} \mid \eta; M \mid \mathsf{if}\ (\mathsf{p}.* = \mathsf{q}.*)\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2$$
$$\mid \mathsf{def}\ X = M_2\ \mathsf{in}\ M_1 \mid X$$

$$\eta ::= \mathsf{p}.e \to \mathsf{q} \mid \mathsf{p} \to \mathsf{q}[l]$$

$$l ::= \mathrm{L} \mid \mathrm{R}$$

$$e ::= \varepsilon \mid * \mid \mathsf{s} \cdot *$$

- only zero-testing $\rightsquigarrow$ termination is decidable (skipping proof. . . )
- only (arbitrary) constant-testing $\rightsquigarrow$ termination is decidable

# *minimality*

*minimal choreographies*

$$M ::= \mathbf{0} \mid \eta; M \mid \text{if } (\mathsf{p}.* = \mathsf{q}.*) \text{ then } M_1 \text{ else } M_2$$
$$\mid \text{def } X = M_2 \text{ in } M_1 \mid X$$

$$\eta ::= \mathsf{p}.e \to \mathsf{q} \mid \mathsf{p} \to \mathsf{q}[l]$$

$$l ::= \mathrm{L} \mid \mathrm{R}$$

$$e ::= \varepsilon \mid * \mid \mathsf{s} \cdot *$$

- selections can be encoded as communications (but...)

# outline

## *what we get*

- sound encoding of partial recursive functions as minimal choreographies

## *what we get*

- sound encoding of partial recursive functions as minimal choreographies
- by embedding into other choreography models $\rightsquigarrow$ sound encoding of partial recursive functions in that model

## *what we get*

- sound encoding of partial recursive functions as minimal choreographies
- by embedding into other choreography models $\rightsquigarrow$ sound encoding of partial recursive functions in that model
- by adding necessary selections (deterministically) $\rightsquigarrow$ sound encoding of partial recursive functions as minimal processes

## *what we get*

- sound encoding of partial recursive functions as minimal choreographies
- by embedding into other choreography models $\rightsquigarrow$ sound encoding of partial recursive functions in that model
- by adding necessary selections (deterministically) $\rightsquigarrow$ sound encoding of partial recursive functions as minimal processes
- by adding necessary selections and embedding into other choreography models $\rightsquigarrow$ sound encoding of partial recursive functions in a process model (in particular, $\pi$-calculus)

# *conclusions*

- turing completeness of minimal choreographies
- minimal set of primitives
- identifies a deadlock-free, turing-complete fragment of $\pi$-calculus
- core language for studying fundamental properties of choreographies

thank you!