formalising choreographic programming

luís cruz-filipe
(joint work with fabrizio montesi & marco peressotti)

department of mathematics and computer science
university of southern denmark

universidade do minho
july 7th, 2022

the goal

long-term

a certified framework for choreographic programming

the goal

long-term

a certified framework for choreographic programming

in this talk

the first steps

- a core choreographic language
- a proof of turing completeness
- a core process calculus
- a proof of the epp theorem

the goal

long-term

a certified framework for choreographic programming

in this talk

the first steps

- a core choreographic language
- a proof of turing completeness
- a core process calculus
- a proof of the epp theorem

very brief summary

initially presented at types'19, publications at itp'21 & ictac'21

choreographic programming, conceptually

what are choreographies?

high-level global specifications of concurrent and distributed systems

a new programming paradigm

implementations for the local endpoints are automatically generated

- guaranteed to be deadlock-free
- guaranted to satisfy the specification

an example

authentication choreography

```
c.credentials --> ip.x;
If ip.(check x)
Then ip --> s[left]; ip --> c[left]; s.token --> c.t
Else ip --> s[right]; ip --> c[right]
```

an example

authentication choreography

```
c.credentials --> ip.x;
If ip.(check x)
Then ip --> s[left]; ip --> c[left]; s.token --> c.t
Else ip --> s[right]; ip --> c[right]
```

local implementations

```
c :  ip!credentials; ip & { left: s?t; right: 0 }
s :  ip & { left: c!token; right: 0 }
ip:  c?x; If (check x) Then (s(+)left; c(+)left)
                       Else (s(+)right; c(+)right)
```

an example

authentication choreography

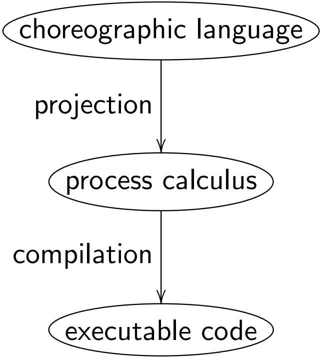
```
c.credentials --> ip.x;
If ip.(check x)
Then ip --> s[left]; ip --> c[left]; s.token --> c.t
Else ip --> s[right]; ip --> c[right]
```

local implementations

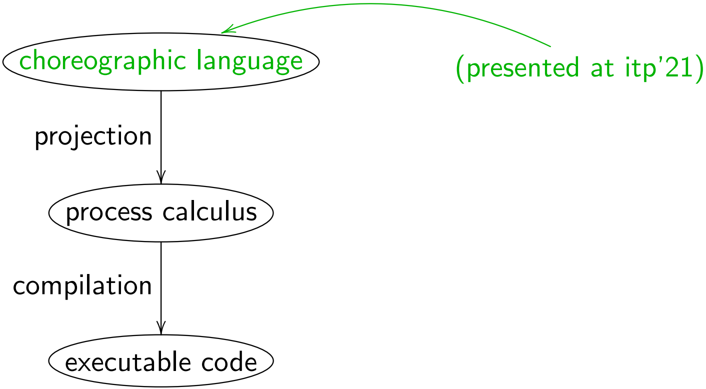
```
c :  ip!credentials; ip & { left: s?t; right: 0 }
s :  ip & { left: c!token; right: 0 }
ip:  c?x; If (check x) Then (s(+)left; c(+)left)
                       Else (s(+)right; c(+)right)
```

(gets tricky in the presence of recursion. . .)

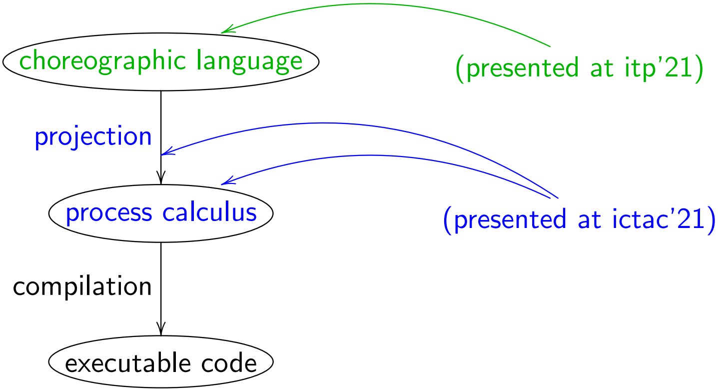
a bird's-eye view



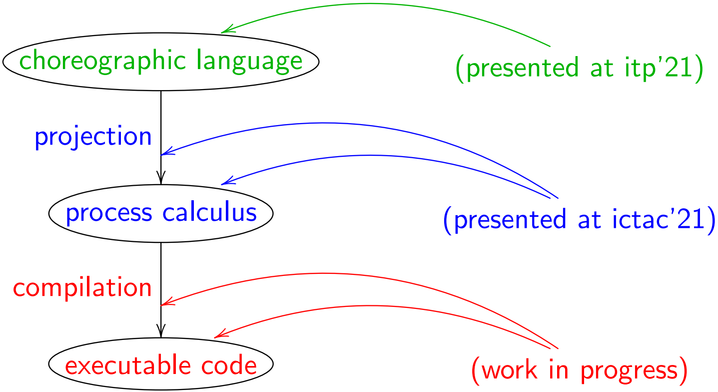
a bird's-eye view



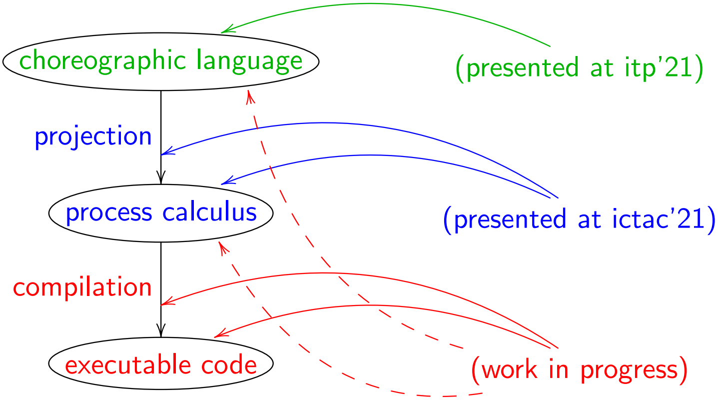
a bird's-eye view



a bird's-eye view



a bird's-eye view



why bother with formalising?

choreographies are a popular topic...

- active research field
- many relevant applications
- potential in choreographic programming

why bother with formalising?

choreographies are a popular topic…

- active research field
- many relevant applications
- potential in choreographic programming

…but there are many disturbing signs

process calculus and session types plagued by wrong proofs

- complex definitions, long proofs by structural induction
- situation pointed out at itp'15
  - formalization of a published journal article
  - most proofs were wrong (but the theorems held)
- big revision of decidability results in the last few years
  - published proofs of both A and $\neg A$ for quite a few A…

our language

a minimal language

- value communication
- label selections (for projection)
- conditionals
- trailing procedure calls (for recursion)

our language

a minimal language

- value communication
- label selections (for projection)
- conditionals
- trailing procedure calls (for recursion)

agnostic language

- parametric on expressions and values
- only two labels

the first step

choreographic language

- syntax and semantics
- progress and deadlock-freedom
- properties of the semantics: determinism, confluence
- turing-completeness from the communication structure

(itp'21)

the first step

choreographic language

- syntax and semantics
- progress and deadlock-freedom
- properties of the semantics: determinism, confluence
- turing-completeness from the communication structure

(itp'21)

methodology

- closely followed a published reference
- formalizing took less time than getting **that** paper accepted
- no wrong proofs found, but. . .

a bit on the process

first attempt: a miserable failure

- bad model of *out-of-order execution*
  `p.e --> q.x; r.e' --> s.y` has two possible reduction paths

a bit on the process

first attempt: a miserable failure

- bad model of *out-of-order execution*
- pen-and-paper definition by means of a structural precongruence (ugh)
- properties are very "intuitive" and never* actually proved
- the number of auxiliary results exploded, with no end in sight

*to the best of the speaker's knowledge

a bit on the process

first attempt: a miserable failure

- bad model of *out-of-order execution*
- pen-and-paper definition by means of a structural precongruence (ugh)
- properties are very "intuitive" and never* actually proved
- the number of auxiliary results exploded, with no end in sight

*to the best of the speaker's knowledge

a weird coincidence?

- oddly enough, this is also where students get stuck

is this good or bad?

second attempt: a success story with side-effects

- model out-of-order execution using an lts
- "intuitive" properties no longer needed (or can be proved)
- auxiliary lemmas disappeared
- final proof of confluence around 25% of the size of the previous (incomplete) development

is this good or bad?

second attempt: a success story with side-effects

- model out-of-order execution using an lts
- "intuitive" properties no longer needed (or can be proved)
- auxiliary lemmas disappeared
- final proof of confluence around 25% of the size of the previous (incomplete) development

and the cherry on top of the cake

our students also liked the new definitions :-)

random thoughts

proof layering

as usual, the theory is developed in "layers", each depending on the previous

- confluence and determinism of the semantics were key ingredients for turing-completeness
- once the "right" definitions were there, the development was very smooth

turing completeness

very classical proof

shown: all partial recursive functions can be implemented as a choreography

- language where values are natural numbers, minimal set of expressions
- a choreography C implements a function $f : \mathbb{N}^n \to \mathbb{N}$ with input processes $p_1, \ldots, p_n$ and output process q if:
  - if $f(k_1, \ldots, k_n)$ is defined and each p_i initially stores k_i, then execution of C terminates in a state where q stores $f(k_1, \ldots, k_n)$
  - if $f(k_1, \ldots, k_n)$ is undefined and each p_i initially stores k_i, then execution of C never terminates

the second step

the epp theorem

- definition of a suitable process calculus
- formalisation of endpoint projection
- challenges: partial functions (branching terms, merging, projection)
- different solutions (dedicated terms, auxiliary types, indirect definitions)
- case explosion (partially) handled by automation

(ictac'21)

the process calculus

networks

finite sets of processes running in parallel

behaviours

local counterparts to the choreography actions

- send and receive
- choice and branching
- conditional
- trailing procedure calls

agnostic language as before

- parametric on expressions and values
- only two labels

compilation, informally

actions split in their components

- value communication $\rightsquigarrow$ send/receive pair
- label selection $\rightsquigarrow$ choice/branching pair
- conditional $\rightsquigarrow$ conditional
- procedure call $\rightsquigarrow$ procedure call

compilation, informally

actions split in their components

- value communication $\rightsquigarrow$ send/receive pair
- label selection $\rightsquigarrow$ choice/branching pair
- conditional $\rightsquigarrow$ conditional
- procedure call $\rightsquigarrow$ procedure call

knowledge of choice

when a process makes a choice, other processes' behaviours can only depend on it after it has been communicated to them

compilation and knowledge of choice

authentication choreography, wrong

```
c.credentials --> ip.x;
If ip.(check x)
Then s.token --> c.t
Else 0
```

local implementations

```
ip:  c?x; If (check x) Then 0 Else 0
c :  ip!credentials; ???
s :  ???
```

compilation and knowledge of choice

authentication choreography, right

```
c.credentials --> ip.x;
If ip.(check x)
Then ip --> s[left]; ip --> c[left]; s.token --> c.t
Else ip --> s[right]; ip --> c[right]
```

local implementations

```
c :  ip!credentials; ip & { left: s?t; right: 0 }
s :  ip & { left: c!token; right: 0 }
ip:  c?x; If (check x) Then (s(+)left; c(+)left)
                       Else (s(+)right; c(+)right)
```

compilation and knowledge of choice

authentication choreography, with logger

```
c.credentials --> ip.x;
If ip.(check x)
Then ip.(x,yes) --> l.y; (...)
Else ip.(x,no) --> l.y; (...)
```

local implementations

```
l :  ip?y
(...)
```

the challenges of partiality

compilation is a partial function

- failure can arise from trying to combine (merge) incompatible branches of a conditional

the challenges of partiality

compilation is a partial function

- failure can arise from trying to combine (merge) incompatible branches of a conditional

all coq functions are total

- explicit terms for failure
- option monad
- proof terms where needed

the challenges of partiality

compilation is a partial function

- failure can arise from trying to combine (merge) incompatible branches of a conditional

all coq functions are total

- explicit terms for failure
  (requires extended syntax, generates isomorphic structures)
- option monad
  (requires a lot of case analysis, horrible proofs)
- proof terms where needed
  (requires bookkeeping, proof irrelevance)

the challenges of partiality

compilation is a partial function

- failure can arise from trying to combine (merge) incompatible branches of a conditional

all coq functions are total

- explicit terms for failure
  (requires extended syntax, generates isomorphic structures)
- option monad
  (requires a lot of case analysis, horrible proofs)
- proof terms where needed
  (requires bookkeeping, proof irrelevance)

$\rightsquigarrow$ no "best" solution, we use a bit of everything

the challenges of case explosion

the root of all problems

the main results require proofs by structural induction, often on two objects

- enormous amounts of cases (e.g. 512, with one subcase further dividing into 64)
- strong similarities, but still slightly different proofs

coq to the rescue!

automation features and tactic language

what's next?

implementation

using coq's extraction mechanism, we can obtain a certified compiler from choreographies to processes

- next step: build an (uncertified?) compiler to a real programming language
- extend the choreographic language (and the process calculus) with other interesting constructs

conclusions

formalising choreographic programming:

- is feasible
- is useful
- can speed up things

conclusions

formalising choreographic programming:

- is feasible
  - we did it (at least partially)
- is useful
  - our theory benefitted from it
- can speed up things
  - convincing coq is faster than convincing reviewers. . .

thank you!