

formalising choreographic programming

luís cruz-filipe

(joint work with fabrizio montesi & marco peressotti)

department of mathematics and computer science
university of southern denmark

ictac 2021
september 8th, 2021

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 core process calculus
- certified choreography compilation

the goal

long-term

a certified framework for choreographic programming

in this talk

the first steps

- a core choreographic language
- a core process calculus
- certified choreography compilation

history

extends work initially presented at types'19 and at itp'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
- guaranteed 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 }
```

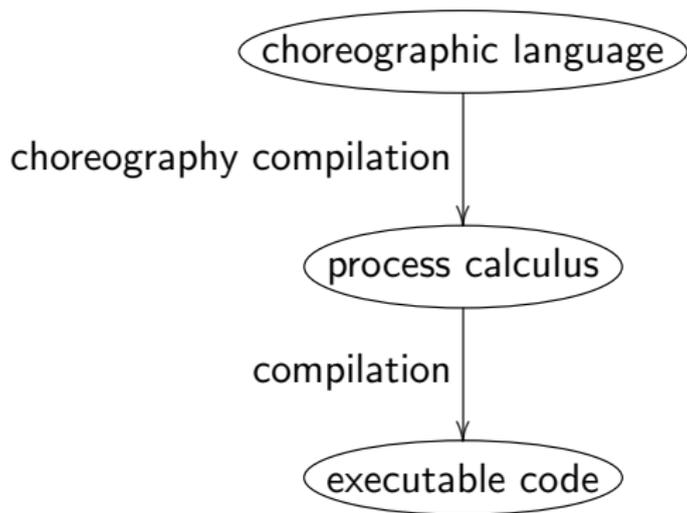
```
s : ip & {left: c!token; right: 0 }
```

```
ip: c?x; If (check x) Then (s(+)left; c(+)left)
```

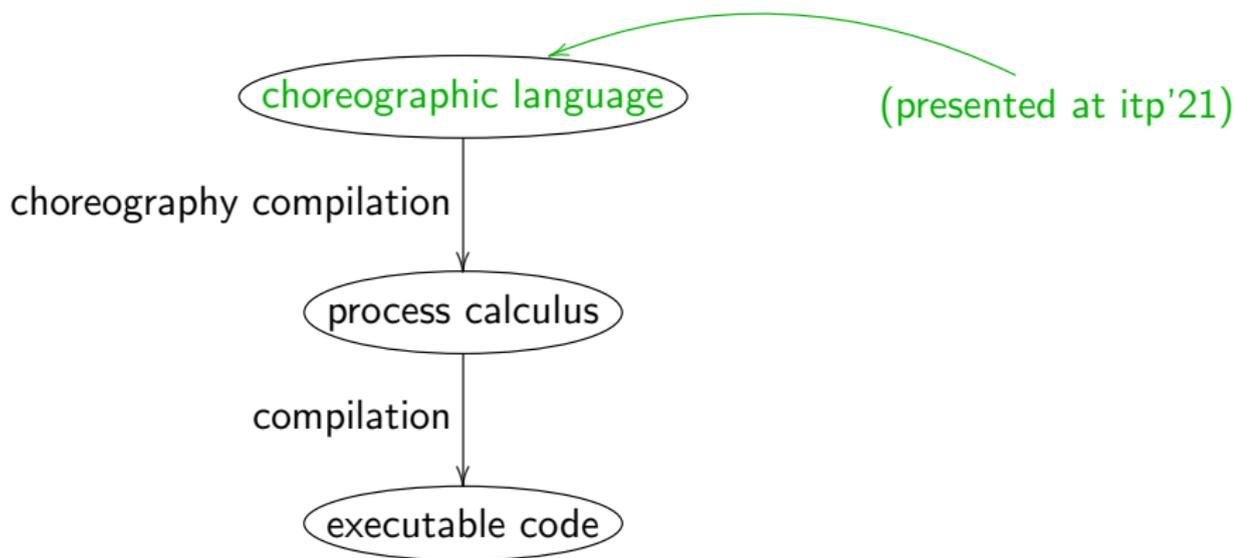
```
Else (s(+)right; c(+)right)
```

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

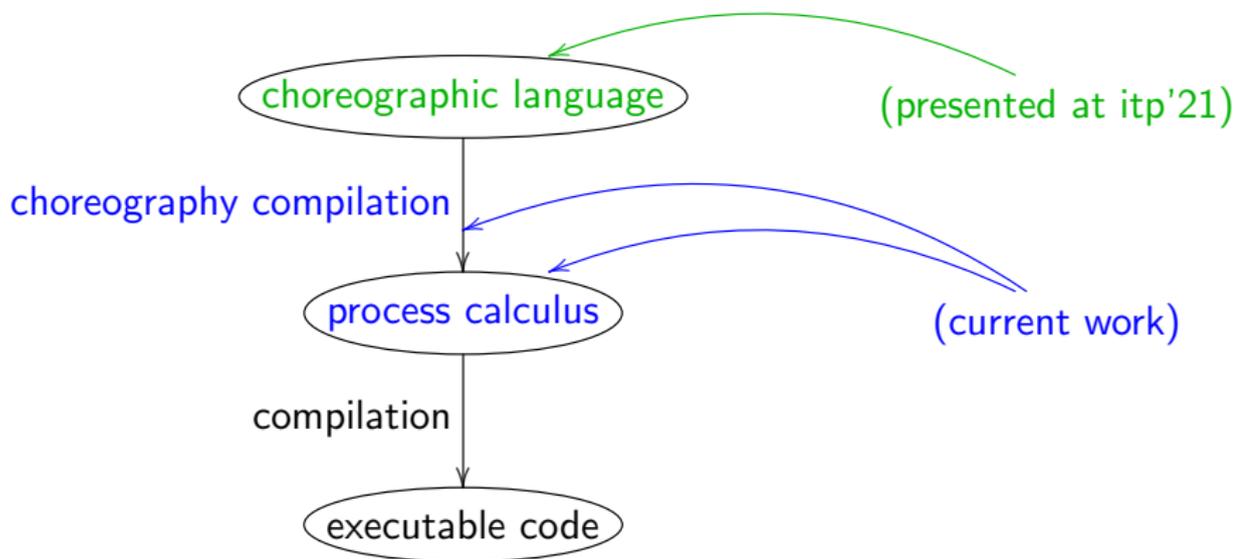
a bird's-eye view



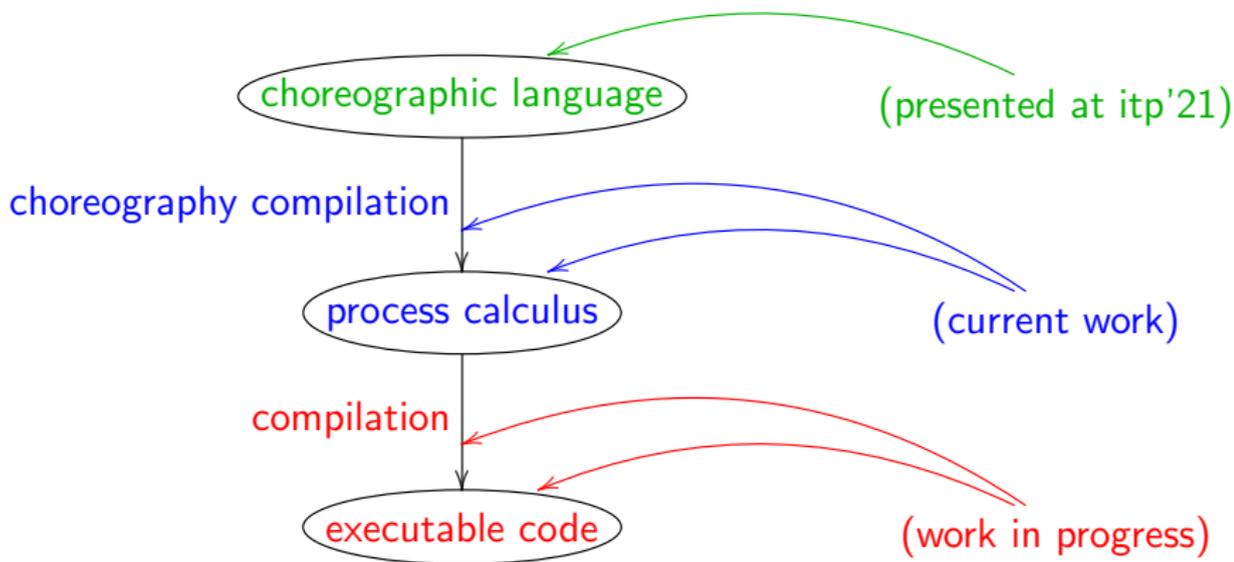
a bird's-eye view



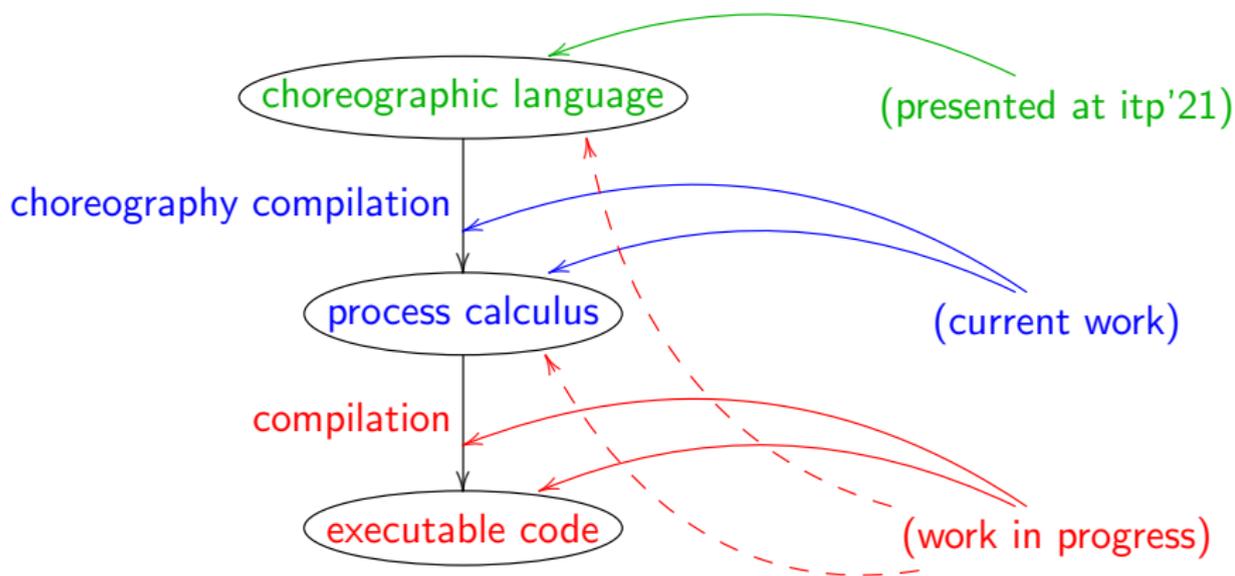
a bird's-eye view



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 ...

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 choreography language

a minimal language

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

the choreography 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 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)

↔ 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

thank you!