formalizing a turing-complete choreographic language in coq

luís cruz-filipe

(joint work with fabrizio montesi & marco peressotti)

department of mathematics and computer science
university of southern denmark

interactive theorem proving
july 1st, 2021

the goal

long-term

a certified framework for choreographic programming

the goal

long-term

a certified framework for choreographic programming

in this work

the first steps

- a core choreographic language
- syntax and semantics
- a proof of turing completeness

the goal

long-term

a certified framework for choreographic programming

in this work

the first steps

- a core choreographic language
- syntax and semantics
- a proof of turing completeness

teaser

some interesting conclusions. . .

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

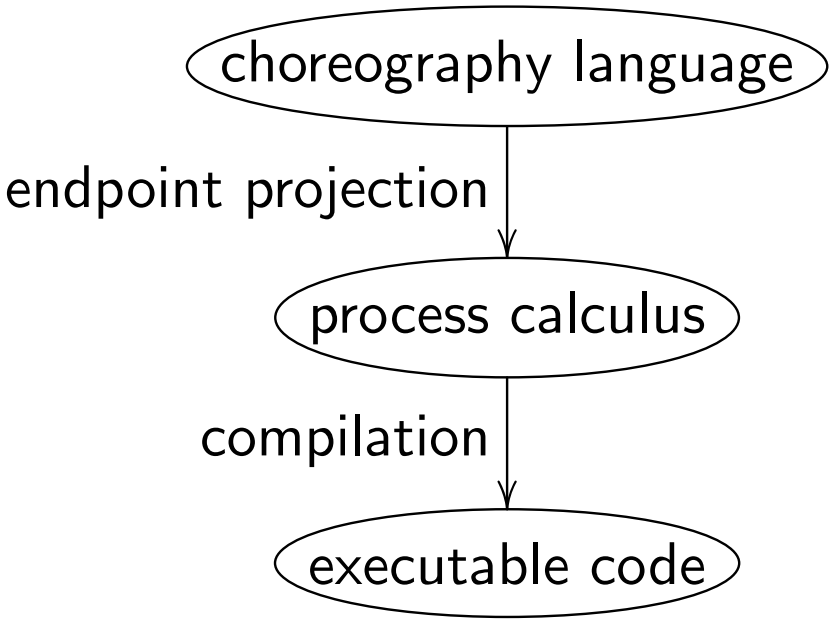
```
c.credentials --> ip.x;
If ip.(check x)
Then ip --> s[left]; ip --> c[left]; s.token --> c.t
Else ip --> s[right]; ip --> c[right]
```

local implementations

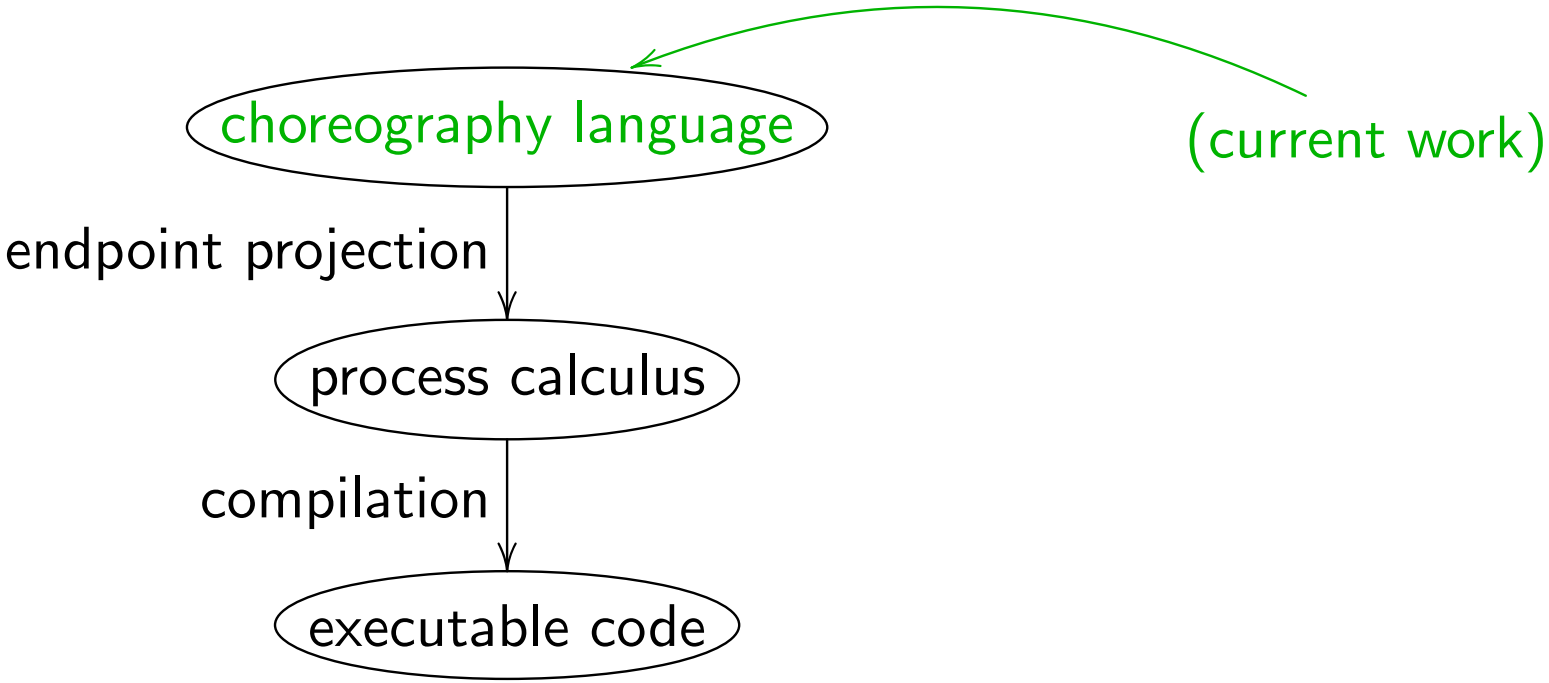
```
c :  ip!credentials; ip & {left:  s?t; right:  0 }
s :  ip & {left:  c!token; right:  0 }
ip:  c?x; If (check x) Then (s(+)left; c(+)left) Else
(s(+)right; c(+)right)
```

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

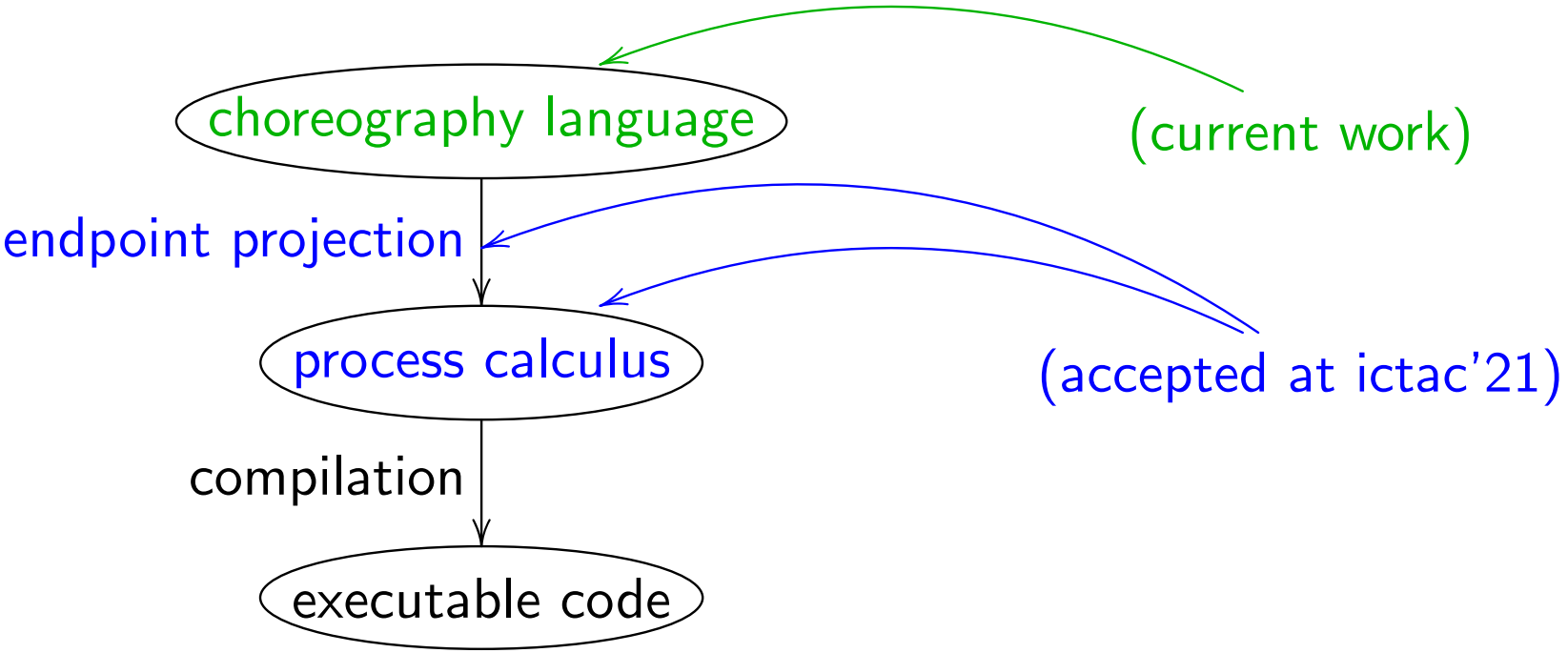
a bird's-eye view



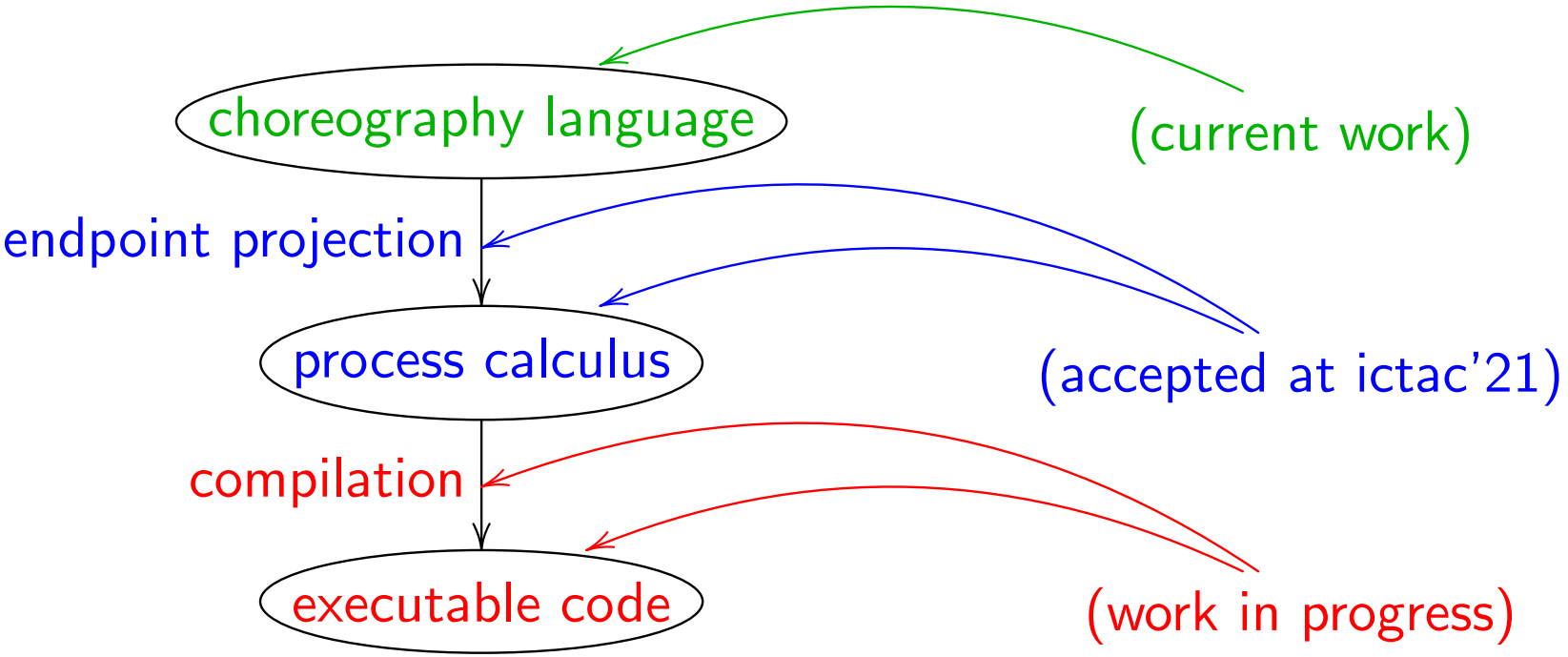
a bird's-eye view



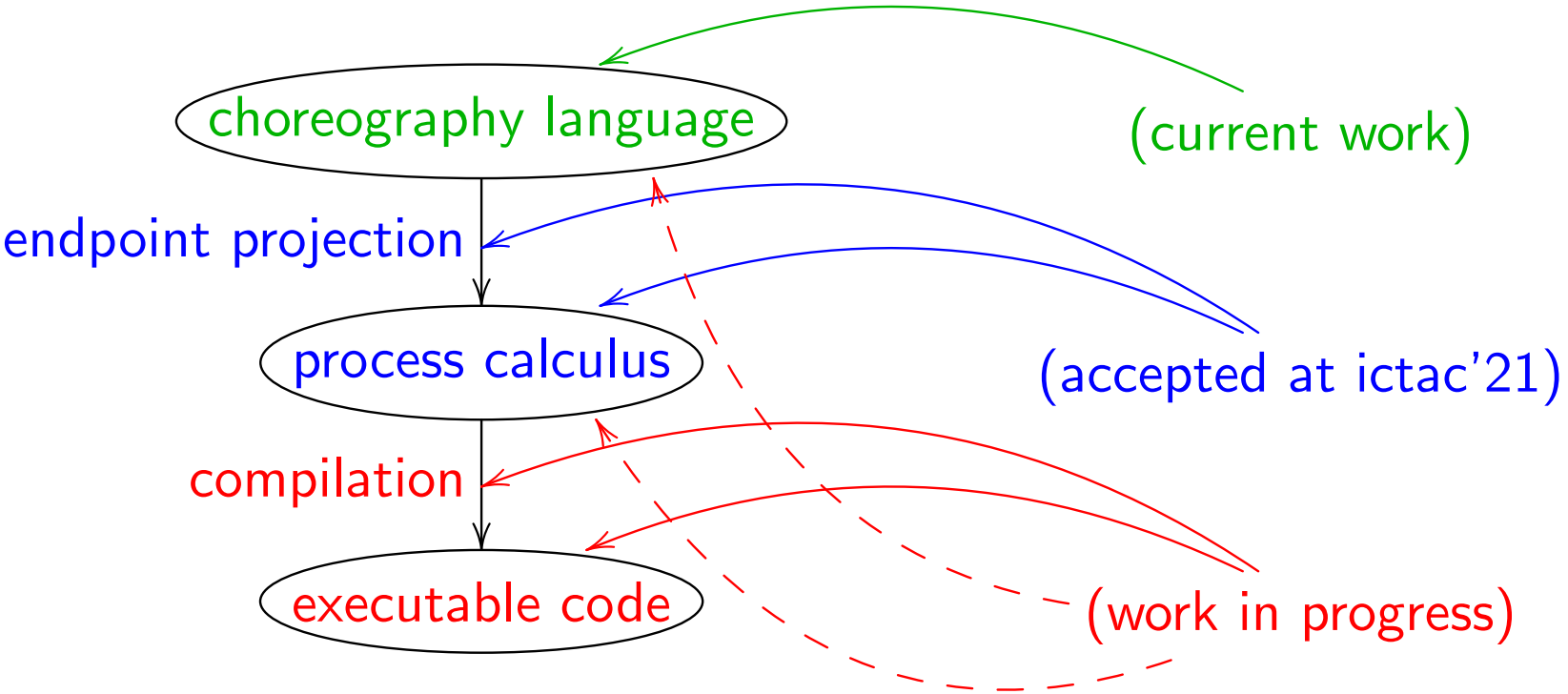
a bird's-eye view



a bird's-eye view



a bird's-eye view



why bother?

choreographies are a popular topic...

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

why bother?

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...

contribution

formalization of a core choreography language

- parametric on expressions, values, &c
- syntax and semantics
- progress and deadlock-freedom
- properties of the semantics: determinism, confluence
- turing-completeness from the communication structure

contribution

formalization of a core choreography language

- parametric on expressions, values, &c
- syntax and semantics
- progress and deadlock-freedom
- properties of the semantics: determinism, confluence
- turing-completeness from the communication structure

methodology

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

is this good or bad?

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

is this good or bad?

first attempt: a miserable failure

- bad model of *out-of-order execution*
- pen-and-paper definition by means of a structural precongruence (ugh)
- the number of auxiliary results exploded, with no end in sight

is this good or bad?

first attempt: a miserable failure

- bad model of *out-of-order execution*
- pen-and-paper definition by means of a structural precongruence (ugh)
- the number of auxiliary results exploded, with no end in sight

two weird coincidences?

- oddly enough, this is also where students get stuck
- properties are very "intuitive" and actually never* proved

*to the best of the speaker's knowledge

is this good or bad? (cont'd)

second attempt: a success story with side-effects

- alternative approach to out-of-order execution (based on the literature)
- "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? (cont'd)

second attempt: a success story with side-effects

- alternative approach to out-of-order execution (based on the literature)
- "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

random thoughts

very classical turing completeness

proved by showing that 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 next steps

endpoint projection

our choreography language can be projected to a particular process calculus language

- both calculus and projection have been formalized in coq
- interesting technical challenges
- see upcoming paper at ictac'21 (available on arXiv)

the next steps

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 choreography language (and the process calculus) with other interesting constructs

conclusions

formalizations of current research:

- are feasible
- are useful
- can speed up things

conclusions

formalizations of current research:

- are feasible
  - we did it
- are useful
  - our theory benefitted from it
- can speed up things
  - convincing the reviewers took three years
  - convincing coq took only two

thank you!