a formalised framework for choreographic programming

luís cruz-filipe

with lovro lugović, fabrizio montesi,
marco peressotti & robert r. rasmussen

department of mathematics and computer science
university of southern denmark

types conference
june 13th, 2023

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

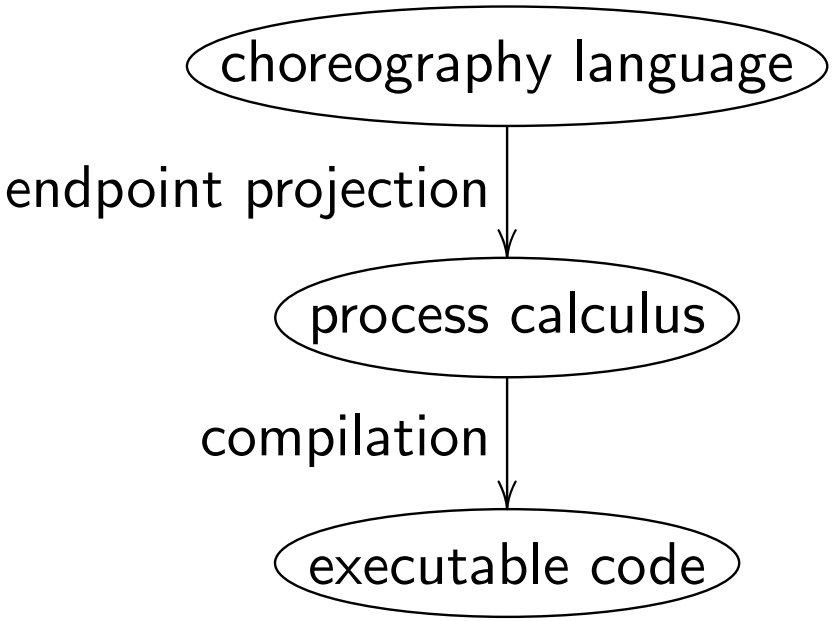
```
c.credentials --> ip.x;
If ip.(check x)
Then ip --> s[left]; ip --> c[left]; s.token --> c.t
Else ip --> s[right]; ip --> c[right]
```

local implementations

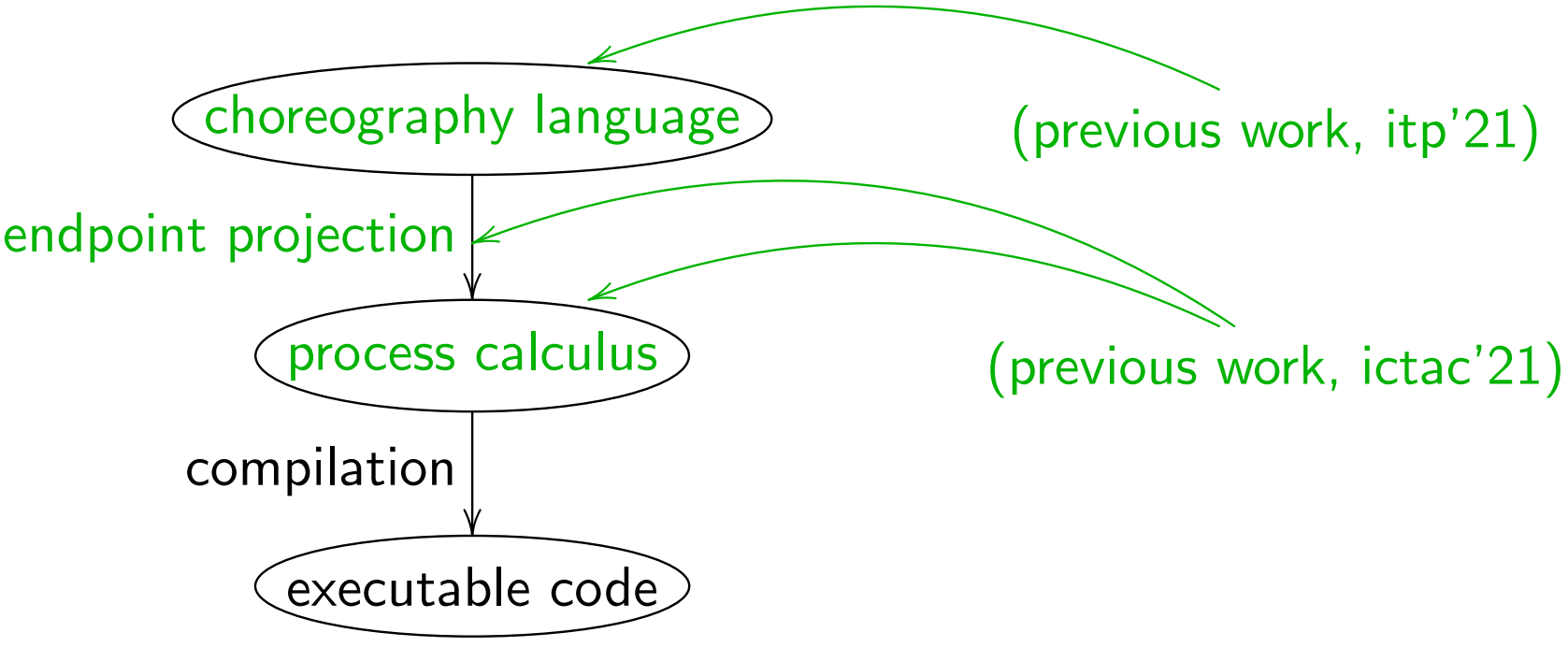
```
c :  ip!credentials; ip & {left:  s?t; right:  0 }
s :  ip & {left:  c!token; right:  0 }
ip:  c?x; If (check x) Then (s(+)left; c(+)left)
                       Else (s(+)right; c(+)right)
```

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

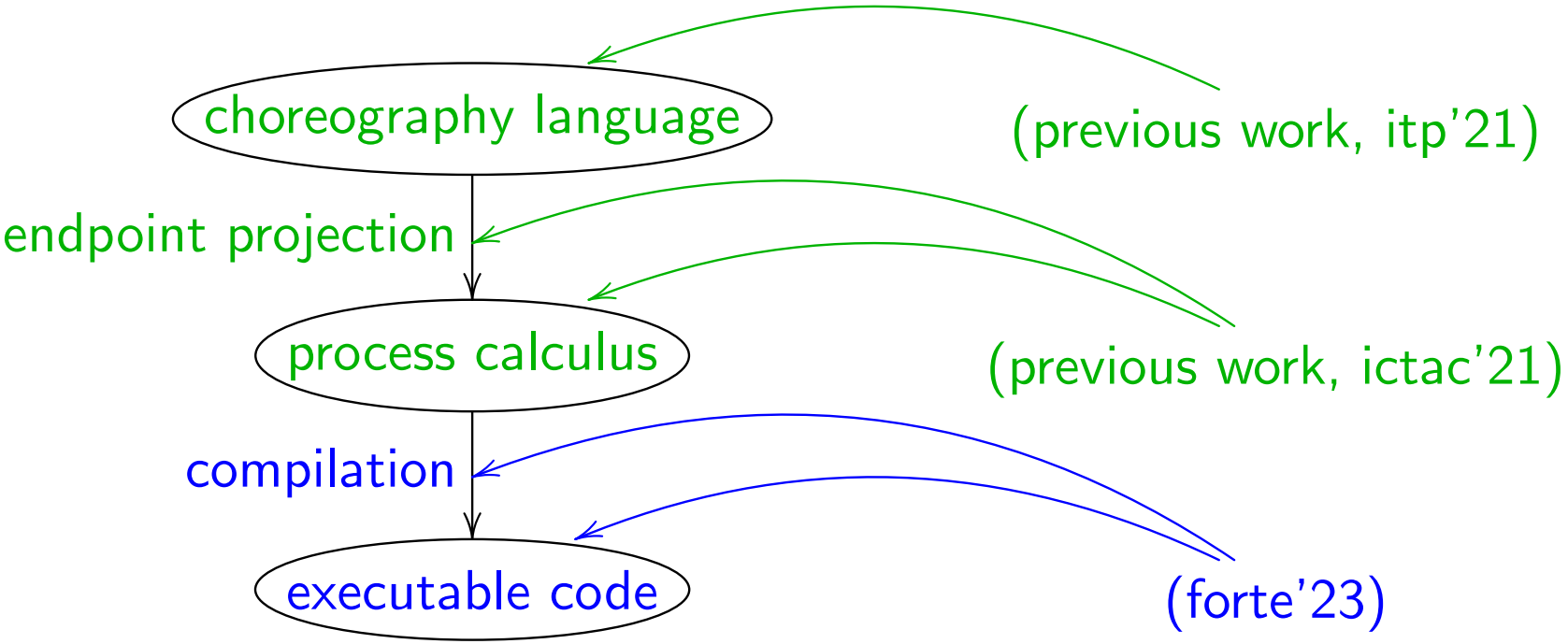
a bird's-eye view



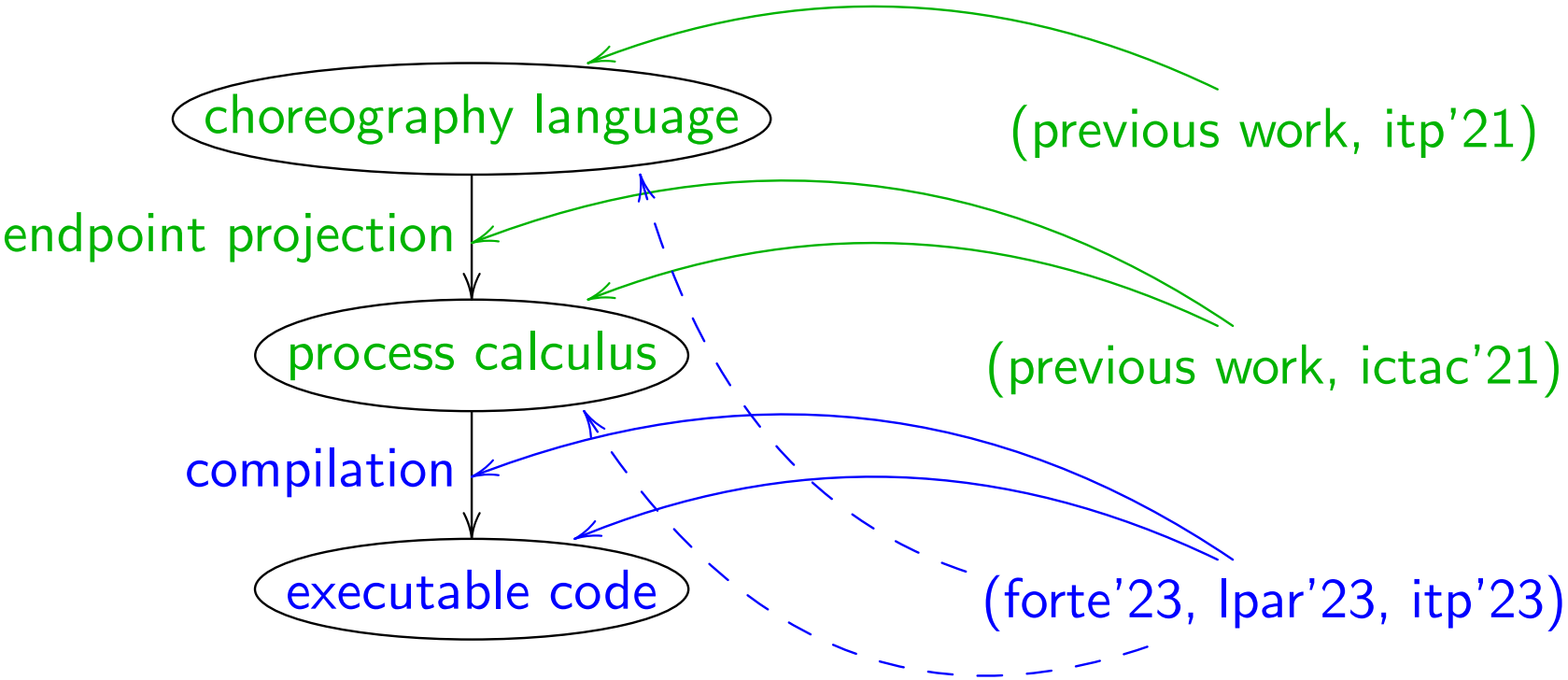
a bird's-eye view



a bird's-eye view



a bird's-eye view



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...

our language

a minimal choreography language

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

our language

a minimal choreography language

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

agnostic language

- parametric on expressions and values
- only two labels

choreographies in coq

choreographic language

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

(itp'21)

choreographies in coq

choreographic language

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

(itp'21)

hard, but insightful

- motivated changes in the presentation of the semantics
- much "cleaner" and more elegant theory
- faster to formalise than to get the original article accepted...

projection

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)

projection

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)

in hindsight...

several steps could be simplified; the current formalisation is significantly shorter than the original one :-)

the next steps

make the framework usable in practice

- build a compiler to a "real" programming language
- identify bottlenecks and extend the theory

the next steps

make the framework usable in practice

- build a compiler to a "real" programming language
- identify bottlenecks and extend the theory

natural candidates

- amendment procedure
- elimination of label selections
- possibility of livelocks (essential for services)

generating executable code

compilation to jolie

- coq's extraction yields a certified implementation of epp
- networks must be translated to a programming language
- structural mapping to jolie
- new feature: annotations in communications

(forte'23)

generating executable code

compilation to jolie

- coq's extraction yields a certified implementation of epp
- networks must be translated to a programming language
- structural mapping to jolie
- new feature: annotations in communications

(forte'23)

to certify or not to certify?

- in principle this compilation could be formalised in coq
- . . . but the benefits of the extra work are less obvious

choreography amendment

authentication choreography, revisited

```
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]
```

- selections can be inferred automatically
- operational correspondence (multi-step, up to permutation)

(itp'23)

choreography amendment

authentication choreography, revisited

```
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]
```

- selections can be inferred automatically
- operational correspondence (multi-step, up to permutation)

(itp'23)

oh, the irony

- the original theorem was wrong
- coq led us to counter-examples and to the correct statement

livelocks and services

authentication, now with retries

```
X = 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]; X
```

livelocks and services

authentication, now with retries

```
X = 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]; X
```

livelocks and services

authentication, now with retries

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

livelocks and services

authentication, now with retries

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

- more permissive projection

  ```
  s :  c!token
  ```

- processes can be muted in recursive calls
- possibility for livelocks

(Ipar'23)

livelocks and services

authentication, now with retries

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

- more permissive projection

  `s : c!token`

- processes can be muted in recursive calls
- possibility for livelocks

(Ipar'23)

coq as a research tool

- a lot of adaptation of old proofs (very doable)
- many technical details that might have been overlooked

closing remarks

conclusions

- a formalised framework for choreographic programming
- compilation to jolie $\rightsquigarrow$ can be used to produce executable code
- robust and modular theory $\rightsquigarrow$ can be used for research

closing remarks

conclusions

- a formalised framework for choreographic programming
- compilation to jolie $\rightsquigarrow$ can be used to produce executable code
- robust and modular theory $\rightsquigarrow$ can be used for research

future work

- proof automation
- formalisation of security protocols
- more features: non-deterministic choice, process spawning

closing remarks

conclusions

- a formalised framework for choreographic programming
- compilation to jolie $\rightsquigarrow$ can be used to produce executable code
- robust and modular theory $\rightsquigarrow$ can be used for research

future work

- proof automation
- formalisation of security protocols
- more features: non-deterministic choice, process spawning

expected challenges

- full-fledged binders
- sublanguages

thank you!