Type-safe interactive web service generation from Scribble

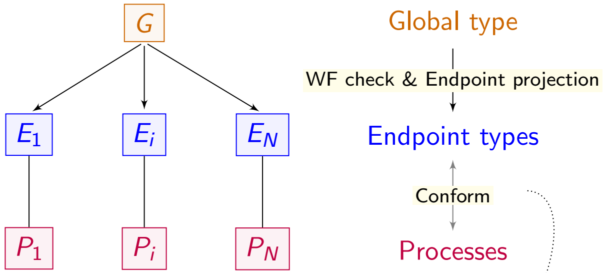
Jonathan King, Nicholas Ng, Nobuko Yoshida

18 Dec 2018 — ABCD project meeting

The project

- **Multiparty** Session Types for interactive web applications
- Scribble applied to PureScript & WebSocket
  - PureScript: strongly-typed functional language, compiles to JavaScript
  - WebSocket: full-duplex communication from the browser
- Embedding of local types/Endpoint FSMs as type classes and constraints
- Jonathan King's final year Master's project
- 8-months of term time work (concurrent with lectures)

Multiparty Session Types (MPST)



Specify

- Global msg-passing protocol

Implement

- Endpoint processes

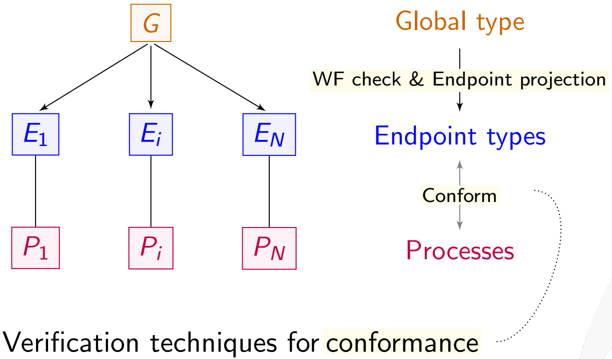
Guarantees

- ✓ Communication safety
- ✓ Deadlock freedom
- ✓ Protocol fidelity

Verification techniques for conformance

- Direct static type checking against implementation
- Runtime monitoring/checks
- APIs/code generation from session types

Multiparty Session Types (MPST)



Specify

- Global msg-passing protocol

Implement

- Endpoint processes

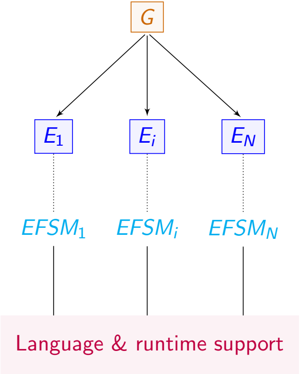
Guarantees

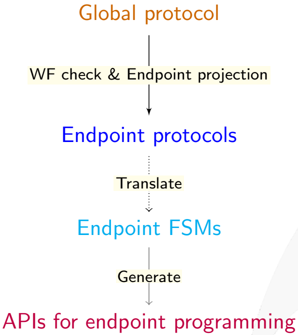
- ✓ Communication safety
- ✓ Deadlock freedom
- ✓ Protocol fidelity

Verification techniques for conformance

- Direct static type checking against implementation
- Runtime monitoring/checks
- APIs/code generation from session types ⇐

Implementations and applications of MPST using Scribble




Global protocol

WF check & Endpoint projection

Endpoint protocols

Translate

Endpoint FSMs

Generate

APIs for endpoint programming

Global Scribble protocol

```
global protocol P(role S, C)
{ Msg(int)   from S to C;
  Reply(bool) from C to S; }
```

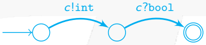
Project

Endpoint Scribble protocol S

```
local protocol P_S(role C)
{ Msg(int)   to C;
  Reply(bool) from C; }
```

Translate

Endpoint FSM for S



Generate/Embed

Endpoint APIs (for users), e.g. Java

```
class S1 { S2 Send(int x) {} }
class S2 { End Recv(boolean b) {} }
```

Implementations and applications of MPST using Scribble

Some example uses:

	Language	Transports
Hybrid session verification (FASE'16) Explicit connection actions (FASE'17)	Java	TCP, SSL/TCP, HTTP
Typestate generation (SCP, 2017)	Java	Java methods
Linear decomposition (ECOOP'17)	Scala	TCP, shared mem., Akka actors
Session Type Provider (CC'18)[1]	F#	TCP
Role-parametric MPST (POPL'19)[2]	Go	TCP, shared mem.

[1] Talk before this

[2] Talk this afternoon

Implementations and applications of MPST using Scribble

Some example uses:

	Language	Transports
Hybrid session verification (FASE'16) Explicit connection actions (FASE'17)	Java	TCP, SSL/TCP, HTTP
Typestate generation (SCP, 2017)	Java	Java methods
Linear decomposition (ECOOP'17)	Scala	TCP, shared mem., Akka actors
Session Type Provider (CC'18)[1]	F#	TCP
Role-parametric MPST (POPL'19)[2]	Go	TCP, shared mem.

- All target desktop/distributed applications
- Can we apply it to web applications?

[1] Talk before this

[2] Talk this afternoon

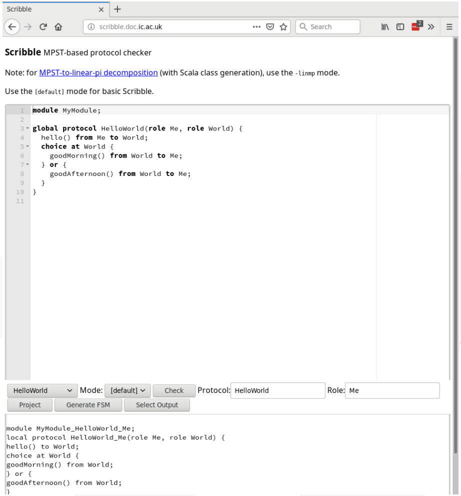
Example: Scribble playground

Web-based interface to the Scribble tool

- **Check** protocols
- **Project** protocol to endpoint protocols
- **Generate FSM** from Scribble protocols

Communicates with server to execute Scribble

```
global protocol Playground(role C, role S) {
  choice at C {
    Verify(String) from C to S; // WF checks
    Result(QResult) from S to C;
  } or {
    Project(String, String, String) from C to S;
    Result(QResult) from S to C;
  } or {
    FSM(String, String, String) from C to S;
    Result(QResult) from S to C;
  }
  do Playground(C , S);
}
```



Scribble MPST-based protocol checker

Note: for MPST-to-linear-pi decomposition (with Scala class generation), use the `-linmp` mode.

Use the `[default]` mode for basic Scribble.

```
module MyModule;

global protocol HelloWorld(role Me, role World) {
  hello() from Me to World;
  choice at World {
    goodMorning() from World to Me;
  } or {
    goodAfternoon() from World to Me;
  }
}
```

HelloWorld Mode: [default] Protocol: HelloWorld Role: Me

```
module MyModule_HelloWorld_Me;
local protocol HelloWorld_Me(role Me, role World) {
hello() to World;
choice at World {
goodMorning() from World;
} or {
goodAfternoon() from World;
}
```

(Similar web-based *playground* exist for Go, Rust, etc.)

Scribble-based API generation for the web

	Language	Transports
Hybrid session verification (FASE'16) Explicit connection actions (FASE'17)	Java	TCP, SSL/TCP, HTTP
Typestate generation (SCP, 2017)	Java	Java methods
Linear decomposition (ECOOP'17)	Scala	TCP, shared mem., Akka actors
Session Type Provider (CC'18)	F#	TCP
Role-parametric MPST (POPL'19)	Go	TCP, shared mem.
This work	**JavaScript**	**WebSocket**

Challenge: JavaScript not statically typed

PureScript

A strongly-typed functional programming language that compiles to JavaScript

From PureScript homepage:

- Compile to readable JavaScript and reuse existing JavaScript code easily
- An extensive collection of libraries for development of web applications, web servers, apps and more
- Excellent tooling and editor support with instant rebuilds
- An active community with many learning resources
- Build real-world applications using functional techniques and expressive types, such as:
    - Algebraic data types and pattern matching
    - Row polymorphism and extensible records
    - Higher kinded types
    - Type classes with functional dependencies
    - Higher-rank polymorphism

PureScript

A strongly-typed functional programming language that compiles to JavaScript

From PureScript homepage:

- Compile to readable JavaScript and reuse existing JavaScript code easily
- An extensive collection of libraries for development of web applications, web servers, apps and more
- Excellent tooling and editor support with instant rebuilds
- An active community with many learning resources
- Build real-world applications using functional techniques and expressive types, such as:
  - Algebraic data types and pattern matching
  - Row polymorphism and extensible records
  - Higher kinded types
  - Type classes with functional dependencies
  - Higher-rank polymorphism

PureScript code generation

PureScript types are generated from the EFSMs, informally:

- Each **state** is a type
- Each **transition** is a type class instance

The (multi-parameter) type classes for each kind of *transition*:

- Send: message send
- Recv: message receive
- Select: selection (receive label)
- Branch: branching (send label)

EFSM transitions as type classes (1)

(Simplified) **Send** and **Receive** type classes, parametrised by s, t, a

$\texttt{class Send } s^{!} \; t \; a \mid s^{!} \rightsquigarrow t \; a$

$\texttt{class Recv } s^{?} \; t \; a \mid s^{?} \rightsquigarrow t \; a$

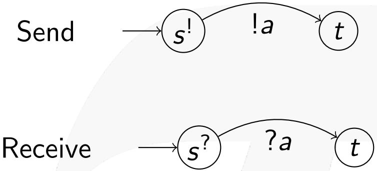
$s^{!}/s^{?}$: Sending/Receiving state

a: Payload type

t: State after transition

Functional dependencies:

(t, a) uniquely identified by $s^{!}/s^{?}$



Send

Receive

EFSM transitions as type classes (2)

(Simplified) **Selection** and **Branching** type classes, parameterised by ts

class Select $s^{!}$ ts | $s^{!} \leadsto ts$

class Branch $s^{?}$ ts | $s^{?} \leadsto ts$

$s^{!}/s^{?}$: Selecting/Branching state

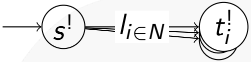
ts: Row list of tuples $(l_i, t_i)_{i \in |ts|}$

l_i, t_i: Branching label i, continuation state i

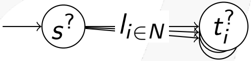
Functional dependencies:

ts uniquely identified by $s^{!}/s^{?}$

Branch (output t_i only)

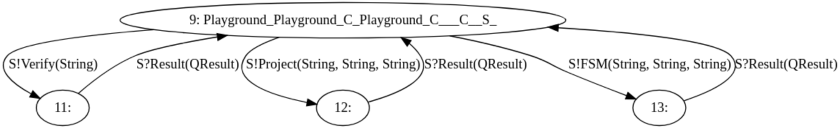


Select (input t_i only)



EFSM as typing constraints

- Constrained by types and *functional dependencies*
- Effectively embedding EFSMs as types: type checks = conformance
- State transitions linearity ensured by Indexed Monad


9: Playground_Playground_C_Playground_C___C__S_
S!Verify(String)
S?Result(QResult)
S!Project(String, String, String)
S?Result(QResult)
S!FSM(String, String, String)
S?Result(QResult)
11:
12:
13:

```
foreign import data S9 :: Type
foreign import data S9Verify :: Type
foreign import data S9Project :: Type
foreign import data S9FSM :: Type
foreign import data S11 :: Type
foreign import data S12 :: Type
foreign import data S13 :: Type
```

```
instance initialClient :: Initial Client S9
instance terminalClient :: Terminal Client Void
instance sendS9Verify :: Send Server S9Verify S11 Verify
instance sendS9Project :: Send Server S9Project S12 Project
instance sendS9FSM :: Send Server S9FSM S13 FSM
instance selectS9 :: Select Server S9 (
                    Cons "project" S9Project (
                    Cons "fsm" S9FSM (
                    Cons "verify" S9Verify Nil)))
instance receiveS11 :: Receive Server S11 S9 Result
instance receiveS12 :: Receive Server S12 S9 Result
instance receiveS13 :: Receive Server S13 S9 Result
```

Scribble playground

Demo

Scribble

Omitted in talk:

- Runtime
- Web framework (Halogen library (?))
- Connection actions
- Error handling and reporting

Summary

Specify global protocol in Scribble

Project global protocol into local protocols

Translate local protocol to EFSM

Generate PureScript type constraints of protocol from EFSM

Write web application endpoint in PureScript

Run type-safe web application

(And we have a new Scribble playground!)