Session Types as a Descriptive Tool for Distributed Protocols

Nobuko Yoshida Raymond Hu

Imperial College London

Scribble

- A practical toolchain based on asynchronous MPST
  - Specify real-world application protocols
  - Implement interoperable endpoint programs in mainstream languages

  - Homepage and tutorial: `http://www.scribble.org/`
  - Scribble-Java source: `https://github.com/scribble/scribble-java`

Hello, world: HTTP (GET)

- Hypertext Transfer Protocol
  - e.g. Web browser (Firefox) fetching a page from a Web server (Apache)
  - Client-server request-response “methods”
    - HTTP/1.1 RFCs 7230–7235 [HTTP]
    - `https://tools.ietf.org/html/rfc7230#section-2.1`

[HTTP1.1] `https://tools.ietf.org/html/rfc7230`, etc.

Protocol specification in Scribble

- Protocol = messages + interactions

```
// Message types
sig <java> "demo.betty16.lec1.httpshort.message.client.Request"
    from "demo/betty16/httpshort/message/Request.java"
    as Request;

sig <java> "demo.betty16.lec1.httpshort.message.server.Response"
    from "demo/betty16/shortvers/message/Response.java"
    as Response;
```

```
global protocol Http(role C, role S) {
  // Interaction structure
  Request  from C to S;
  Response from S to C;
}
```

Client implementation in Java

- For now, assume a basic fluent (call-chaining) Java API over TCP sockets

```
String host = "www.doc.ic.ac.uk"; int port = 80;
Buf<Response> buf = new Buf<>();

 c.send(S, new Request("/~rhu/", "1.1", host))
  .receive(S, Response, buf); // Received message read into buf

 c.send(S, new Response("1.1", "..body.."))
  .receive(S, Response, buf);

 c.send(S, new Request("/~rhu/", "1.1", host))
  .send(S, new Request("/~rhu/", "1.1", host))
  .receive(S, Response, buf);
```

- So.. is that it? To implement a good HTTP client program

Client implementation in Java

- For now, assume a basic fluent (call-chaining) Java API over TCP sockets

```
String host = "www.doc.ic.ac.uk"; int port = 80;
Buf<Response> buf = new Buf<>();

c.send(S, new Request("/~rhu/", "1.1", host))
 .receive(S, Response, buf); // Received message read into buf

c.send(S, new Response("1.1", "..body.."))
 .receive(S, Response, buf);
```

The method send(S, Request) ... for the arguments (S, Response)

```
c.send(S, new Request("/~rhu/", "1.1", host))
 .send(S, new Request("/~rhu/", "1.1", host))
 .receive(S, Response, buf);
```

- So.. is that it? To implement a good HTTP client program

Client implementation in Java

- For now, assume a basic fluent (call-chaining) Java API over TCP sockets

```
String host = "www.doc.ic.ac.uk"; int port = 80;
Buf<Response> buf = new Buf<>();

 c.send(S, new Request("/~rhu/", "1.1", host))
  .receive(S, Response, buf); // Received message read into buf

 c.send(S, new Response("1.1", "..body.."))
  .receive(S, Response, buf);

 c.send(S, new Request("/~rhu/", "1.1", host))
  .send(S, new Request("/~rhu/", "1.1", host))
  .receive(S, Response, buf);
```

The method send(S, Request) is undefined for the type Http_C_2

- So.. is that it? To implement a good HTTP client program

Client implementation in Java

- For now, assume a basic fluent (call-chaining) Java API over TCP sockets

```
String host = "www.doc.ic.ac.uk"; int port = 80;
Buf<Response> buf = new Buf<>();

 c.send(S, new Request("/~rhu/", "1.1", host))
  .receive(S, Response, buf); // Received message read into buf

 c.send(S, new Response("1.1", "..body.."))
  .receive(S, Response, buf);

 c.send(S, new Request("/~rhu/", "1.1", host))
  .send(S, new Request("/~rhu/", "1.1", host))
  .receive(S, Response, buf);
```

- So.. is that it? To implement a good HTTP client program

Message types vs. interaction structure

- Simple interaction structure..
  - ..means more work is done in message serialization/deserialization
    - `https://tools.ietf.org/html/rfc7230#section-3`
  - The call-response pattern and top-level data types are checked.. how about serialization/deserializaton?
- Practical protocol specifications: Interplay between data types and interaction structure
  - E.g., can leverage session types to also capture the data protocol
    - *From Data Types to Session Types*:
      A Basis for Concurrency and Distribution (ABCD)
      University of Edinburgh, University of Glasgow, Imperial College London
      `https://groups.inf.ed.ac.uk/abcd/`

HTTP client-server conversation

- `telnet www.doc.ic.ac.uk 80`

```
GET /~rhu/ HTTP/1.1
Host: www.doc.ic.ac.uk
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:47.0) Gecko/20
100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
```

⋮

HTTP client-server conversation

- telnet www.doc.ic.ac.uk 80

```
⋮

HTTP/1.1 200 OK
Date: Mon, 13 Jun 2016 19:42:34 GMT
Server: Apache
Strict-Transport-Security: max-age=31536000; preload; includeSubDomains
Strict-Transport-Security: max-age=31536000; preload; includeSubDomains
Last-Modified: Thu, 14 Apr 2016 12:46:24 GMT
ETag: "74a-53071482f6e0f"
Accept-Ranges: bytes
Content-Length: 1866
Vary: Accept-Encoding
Content-Type: text/html
Via: 1.1 www.doc.ic.ac.uk
```

Decomposing message structures..

- https://github.com/rhu1/scribble-java/tree/rhu1-research/modules/core/src/test/scrib/demo/betty16/lec1/httplong

- Client messages

```
sig <java> "...message.client.RequestLine" from "...message/RequestLine.java"
    as RequestLine; // GET /~rhu/ HTTP/1.1
sig <java> "...message.client.Host" from "...message/Host.java"
    as Host;          // host: www.doc.ic.ac.uk
sig <java> "...message.client.UserAgent" from "...message/UserAgent.java"
    as UserAgent;    // User-Agent: Mozilla/5.0 ... Firefox/38.0
...
```

- Server messages

```
sig <java> "...message.server.HttpVersion" from "...message/HttpVersion.java"
    as HTTPV; // HTTP/1.1
sig <java> "...message.server._200" from "...message/_200.java"
    as 200;   // 200 OK
sig <java> "...message.server._404" from "...message/_404.java"
    as 404;   // 404 Not found
...
```

..promotes more fine-grained interaction structures

```
global protocol Http(role C, role S) {
  do Request(C, S);
  do Response(C, S);
}

global protocol Request(role C, role S) {
  RequestLine from C to S;   // GET /~rhu/ HTTP/1.1
  rec X {
    choice at C {
      Host from C to S;       // Host: www.doc.ic.ac.uk
      continue X;
    } or {
      UserAgent from C to S; // User-Agent: Mozilla/5.0 ...
      continue X;
    } or {
      ...
    } or {
      Body from C to S;
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Http(role C, role S) {
  do Request(C, S);
  do Response(C, S);
}

global protocol Request(role C, role S) {
  RequestLine from C to S;   // GET /~rhu/ HTTP/1.1
  rec X {
    choice at C {
      Host from C to S;      // Host: www.doc.ic.ac.uk
      continue X;
    } or {
      UserAgent from C to S; // User-Agent: Mozilla/5.0 ...
      continue X;
    } or {
      ...
    } or {
      Body from C to S;
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Http(role C, role S) {
  do Request(C, S);
  do Response(C, S);
}

global protocol Request(role C, role S) {
  RequestLine from C to S;   // GET /~rhu/ HTTP/1.1
  rec X {
    choice at C {
      Host from C to S;       // Host: www.doc.ic.ac.uk
      continue X;
    } or {
      UserAgent from C to S; // User-Agent: Mozilla/5.0 ...
      continue X;
    } or {
      ...
    } or {
      Body from C to S;
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Http(role C, role S) {
  do Request(C, S);
  do Response(C, S);
}

global protocol Request(role C, role S) {
  RequestLine from C to S;   // GET /~rhu/ HTTP/1.1
  rec X {
    choice at C {
      Host from C to S;       // Host: www.doc.ic.ac.uk
      continue X;
    } or {
      UserAgent from C to S; // User-Agent: Mozilla/5.0 ...
      continue X;
    } or {
      ...
    } or {
      Body from C to S;
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Http(role C, role S) {
  do Request(C, S);
  do Response(C, S);
}

global protocol Request(role C, role S) {
  RequestLine from C to S;   // GET /~rhu/ HTTP/1.1
  rec X {
    choice at C {
      Host from C to S;       // Host: www.doc.ic.ac.uk
      continue X;
    } or {
      UserAgent from C to S; // User-Agent: Mozilla/5.0 ...
      continue X;
    } or {
      ...
    } or {
      Body from C to S;
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Http(role C, role S) {
  do Request(C, S);
  do Response(C, S);
}

global protocol Request(role C, role S) {
  RequestLine from C to S;   // GET /~rhu/ HTTP/1.1
  rec X {
    choice at C {
      Host from C to S;        // Host: www.doc.ic.ac.uk
      continue X;
    } or {
      UserAgent from C to S; // User-Agent: Mozilla/5.0 ...
      continue X;
    } or {
      ...
    } or {
      Body from C to S;
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Http(role C, role S) {
  do Request(C, S);
  do Response(C, S);
}

global protocol Request(role C, role S) {
  RequestLine from C to S;   // GET /~rhu/ HTTP/1.1
  rec X {
    choice at C {
      Host from C to S;       // Host: www.doc.ic.ac.uk
      continue X;
    } or {
      UserAgent from C to S; // User-Agent: Mozilla/5.0 ...
      continue X;
    } or {
      ...
    } or {
      Body from C to S;
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Reponse(role C, role S) {
  HttpVers from S to C; // HTTP/1.1
  choice at S {
    200 from S to C;     // 200 OK
  } or {
    404 from S to C;     // 404 Not found
  } or {
    ...
  }

  rec Y {
    choice at S {
      Date from S to C;   // Date: Sun, 24 May 2015 21:04:36 GMT
      continue Y;
    } or {
      Server from S to C; // Server: Apache
      continue Y;
    } or {
      ...
    } or {
      Body from S to C;   // <html>...</html>
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Reponse(role C, role S) {
  HttpVers from S to C;  // HTTP/1.1
  choice at S {
    200 from S to C;     // 200 OK
  } or {
    404 from S to C;     // 404 Not found
  } or {
    ...
  }

  rec Y {
    choice at S {
      Date from S to C;   // Date: Sun, 24 May 2015 21:04:36 GMT
      continue Y;
    } or {
      Server from S to C; // Server: Apache
      continue Y;
    } or {
      ...
    } or {
      Body from S to C;   // <html>...</html>
    }
} }
```

..promotes more fine-grained interaction structures

```
global protocol Reponse(role C, role S) {
  HttpVers from S to C;  // HTTP/1.1
  choice at S {
    200 from S to C;     // 200 OK
  } or {
    404 from S to C;     // 404 Not found
  } or {
    ...
  }

  rec Y {
    choice at S {
      Date from S to C;   // Date: Sun, 24 May 2015 21:04:36 GMT
      continue Y;
    } or {
      Server from S to C; // Server: Apache
      continue Y;
    } or {
      ...
    } or {
      Body from S to C;   // <html>...</html>
    }
} }
```

Revised client code

```
response(request(new Http_C_1(client), "www.doc.ic.ac.uk"));
```

```
Http_C_3 request(Http_C_1 c1, String host) throws ... {
  return
    c1.send(S, new RequestLine("/~rhu/", "1.1"))
      .send(S, new Host(host))
      .send(S, new Body(""));
}
```

✓ Formatting of request message (request line, fields) is now checked

Revised client code

```
response(request(new Http_C_1(client), "www.doc.ic.ac.uk"));
```

```
Http_C_3 request(Http_C_1 c1, String host) throws ... {
  return
    c1.send(S, new RequestLine("/~rhu/", "1.1"))
      .send(S, new Host(host))
      .send(S, new Body(""));
}
```

✓ Formatting of request message (request line, fields) is now checked

Revised client code

```
response(request(new Http_C_1(client), "www.doc.ic.ac.uk"));
```

```
Http_C_3 request(Http_C_1 c1, String host) throws ... {
  return
    c1.send(S, new RequestLine("/~rhu/", "1.1"))
      .send(S, new Host(host))
      .send(S, new Body(""));
}
```

✓ Formatting of request message (request line, fields) is now checked

Revised client code

```
response(request(new Http_C_1(client), "www.doc.ic.ac.uk"));
```

```
void response(Http_C_3 c3) throws ... {
  Http_C_4_Cases status = c3.async(S, HTTPV).branch(S);
  switch (status.op) {
    case _200: responseAux(status.receive(_200)); break;
    case _404: responseAux(status.receive(_404)); break;
    default: throw new RuntimeException("[TODO]: " + status.op);
} }

void responseAux(Http_C_5 c5) throws ... {
  Http_C_5_Cases cases = c5.branch(S);
  switch (cases.op) {
    case DATE:   responseAux(cases.receive(DATE));   break;
    case SERVER: responseAux(cases.receive(SERVER)); break;
    ...
    case BODY: { Buf<Body> buf_body = new Buf<>();
                 cases.receive(BODY, buf_body);
                 System.out.println(buf_body.val.getBody());
                 return; }
    default: throw new RuntimeException("[TODO]: " + cases.op);
} }
```

Revised client code

```
response(request(new Http_C_1(client), "www.doc.ic.ac.uk"));
```

```
void response(Http_C_3 c3) throws ... {
  Http_C_4_Cases status = c3.async(S, HTTPV).branch(S);
  switch (status.op) {
    case _200: responseAux(status.receive(_200)); break;
    case _404: responseAux(status.receive(_404)); break;
    default: throw new RuntimeException("[TODO]: " + status.op);
} }

void responseAux(Http_C_5 c5) throws ... {
  Http_C_5_Cases cases = c5.branch(S);
  switch (cases.op) {
    case DATE:   responseAux(cases.receive(DATE));   break;
    case SERVER: responseAux(cases.receive(SERVER)); break;
    ...
    case BODY: { Buf<Body> buf_body = new Buf<>();
                 cases.receive(BODY, buf_body);
                 System.out.println(buf_body.val.getBody());
                 return; }
    default: throw new RuntimeException("[TODO]: " + cases.op);
} }
```

Revised client code

```
response(request(new Http_C_1(client), "www.doc.ic.ac.uk"));
```

```
void response(Http_C_3 c3) throws ... {
  Http_C_4_Cases status = c3.async(S, HTTPV).branch(S);
  switch (status.op) {
    case _200: responseAux(status.receive(_200)); break;
    case _404: responseAux(status.receive(_404)); break;
    default: throw new RuntimeException("[TODO]: " + status.op);
} }

void responseAux(Http_C_5 c5) throws ... {
  Http_C_5_Cases cases = c5.branch(S);
  switch (cases.op) {
    case DATE:   responseAux(cases.receive(DATE));   break;
    case SERVER: responseAux(cases.receive(SERVER)); break;
    ...
    case BODY: { Buf<Body> buf_body = new Buf<>();
                 cases.receive(BODY, buf_body);
                 System.out.println(buf_body.val.getBody());
                 return; }
    default: throw new RuntimeException("[TODO]: " + cases.op);
} }
```

Revised client code

```
response(request(new Http_C_1(client), "www.doc.ic.ac.uk"));
```

```
void response(Http_C_3 c3) throws ... {
  Http_C_4_Cases status = c3.async(S, HTTPV).branch(S);
  switch (status.op) {
    case _200: responseAux(status.receive(_200)); break;
    case _404: responseAux(status.receive(_404)); break;
    default: throw new RuntimeException("[TODO]: " + status.op);
} }

void responseAux(Http_C_5 c5) throws ... {
  Http_C_5_Cases cases = c5.branch(S);
  switch (cases.op) {
    case DATE:   responseAux(cases.receive(DATE));   break;
    case SERVER: responseAux(cases.receive(SERVER)); break;
    ...
    case BODY: { Buf<Body> buf_body = new Buf<>();
                 cases.receive(BODY, buf_body);
                 System.out.println(buf_body.val.getBody());
                 return; }
    default: throw new RuntimeException("[TODO]: " + cases.op);
} }
```

Revised client code

```
response(request(new Http_C_1(client), "www.doc.ic.ac.uk"));
```

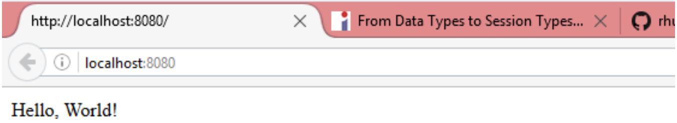
```
void response(Http_C_3 c3) throws ... {
  Http_C_4_Cases status = c3.async(S, HTTPV).branch(S);
  switch (status.op) {
    case _200: responseAux(status.receive(_200)); break;
    case _404: responseAux(status.receive(_404)); break;
    default: throw new RuntimeException("[TODO]: " + status.op);
} }

void responseAux(Http_C_5 c5) throws ... {
  Http_C_5_Cases cases = c5.branch(S);
  switch (cases.op) {
    case DATE:   responseAux(cases.receive(DATE));   break;
    case SERVER: responseAux(cases.receive(SERVER)); break;
    ...
    case BODY: { Buf<Body> buf_body = new Buf<>();
                 cases.receive(BODY, buf_body);
                 System.out.println(buf_body.val.getBody());
                 return; }
    default: throw new RuntimeException("[TODO]: " + cases.op);
} }
```

Hello, world: HTTP (GET)

- Rigorous specification and verification of protocols is important
  (Even for a "simple" two-party call-return)

- Further alternative specifications?
  - Most simplified?
  - Most detailed?

- Similarly for the server
  - Each Scribble client/server is interoperable with (compliant) real-world implementations..
  - ..and with each other

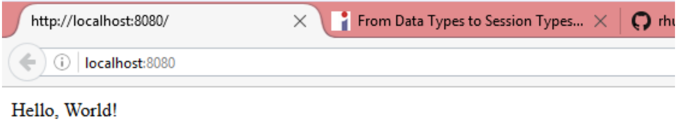


Hello, world: HTTP (GET)

- Rigorous specification and verification of protocols is important
  (Even for a "simple" two-party call-return)

- Further alternative specifications?
  - Most simplified? call-return of monolithic ASCII strings
  - Most detailed?

- Similarly for the server
  - Each Scribble client/server is interoperable with (compliant) real-world implementations..
  - ..and with each other

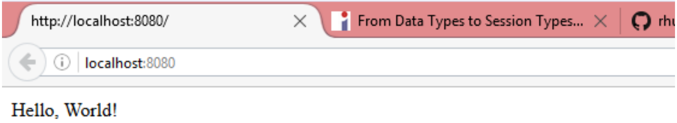


Hello, world: HTTP (GET)

- Rigorous specification and verification of protocols is important
  (Even for a “simple” two-party call-return)

- Further alternative specifications?
  - Most simplified? call-return of monolithic ASCII strings
  - Most detailed? towards “character-perfect” I/O?

- Similarly for the server
  - Each Scribble client/server is interoperable with (compliant) real-world implementations..
  - ..and with each other



The Scribble toolchain

- MPST-based protocol verification
  - Leverage syntactic characteristics of MPST for explicit verification of distributed protocols
    - More expressive protocol language (required for practical applications)
    - Uniform method for integrating MPST features for implementation
- Type-driven code generation (“State Channel” APIs)
  - Type generation: static typing of protocol- and role- specific types
  - Code generation: safety mechanisms and correctness by construction

[Home] Homepage and tutorials. `http://www.scribble.org/`

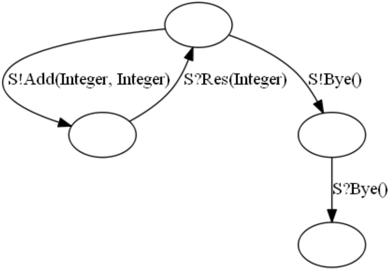
[GitHub] Scribble-Java source code. `https://github.com/scribble/scribble-java`

[FASE16] *Hybrid Session Verification through Endpoint API Generation.* Hu and Yoshida.

[FASE17] *Explicit Connection Actions in Multiparty Session Types.* Hu and Yoshida.

MPST-based protocol verification

```
global protocol Adder(role C, role S) {
  choice at C {
    Add(Integer, Integer) from C to S;
    Res(Integer) from S to C;
    do Adder(C, S);
  } or {
    Bye() from C to S;
    Bye() from S to C;
  }
}
```

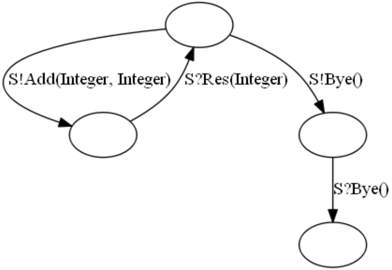


- "Endpoint FSM" characteristics dervied from syntactic MPST:
  - "Well-formed" global protocol: type syntax and projection
  - "Consistent" and deterministic MP choices, no mixed states, ...
- (1-)bounded verification of CFSM system: MPST safety and progress
  - Reception errors, deadlocks, orphan messages, role progress, ...

[FASE17] *Explicit Connection Actions in Multiparty Session Types*. Hu and Yoshida.
[CONCUR15] *Meeting Deadlines Together*. Bocchi, Lange and Yoshida.

MPST-based protocol verification

```
global protocol Adder(role C, role S) {
  choice at C {
    Add(Integer, Integer) from C to S;
    Res(Integer) from S to C;
    do Adder(C, S);
  } or {
    Bye() from C to S;
    Bye() from S to C;
  }
}
```

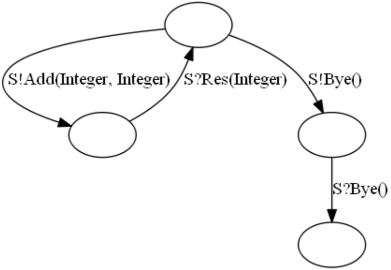


- "Endpoint FSM" characteristics dervied from syntactic MPST:
  - "Well-formed" global protocol: type syntax and projection
  - "Consistent" and deterministic MP choices, no mixed states, ...
- (1-)bounded verification of CFSM system: MPST safety and progress
  - Reception errors, deadlocks, orphan messages, role progress, ...

[FASE17] *Explicit Connection Actions in Multiparty Session Types*. Hu and Yoshida.
[CONCUR15] *Meeting Deadlines Together*. Bocchi, Lange and Yoshida.

MPST-based protocol verification

```
global protocol Adder(role C, role S) {
  choice at C {
    Add(Integer, Integer) from C to S;
    Res(Integer) from S to C;
    do Adder(C, S);
  } or {
    Bye() from C to S;
    Bye() from S to C;
  }
}
```



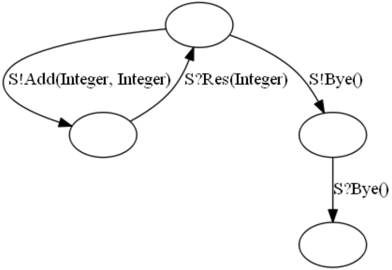
- “Endpoint FSM” characteristics dervied from syntactic MPST:
  - “Well-formed” global protocol: type syntax and projection
  - “Consistent” and deterministic MP choices, no mixed states, ...
- (1-)bounded verification of CFSM system: MPST safety and progress
  - Reception errors, deadlocks, orphan messages, role progress, ...

[FASE17] *Explicit Connection Actions in Multiparty Session Types*. Hu and Yoshida.

[CONCUR15] *Meeting Deadlines Together*. Bocchi, Lange and Yoshida.

MPST-based protocol verification

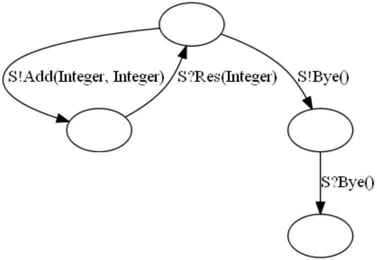
```
global protocol Adder(role C, role S) {
  choice at C {
    Add(Integer, Integer) from C to S;
    Res(Integer) from S to C;
    do Adder(C, S);
  } or {
    Bye() from C to S;
    Bye() from S to C;
  }
}
```



- "Endpoint FSM" characteristics dervied from syntactic MPST:
  - "Well-formed" global protocol: type syntax and projection
  - "Consistent" and deterministic MP choices, no mixed states, ...
- (1-)bounded verification of CFSM system: MPST safety and progress
  - Reception errors, deadlocks, orphan messages, role progress, ...

[FASE17] *Explicit Connection Actions in Multiparty Session Types*. Hu and Yoshida.
[CONCUR15] *Meeting Deadlines Together*. Bocchi, Lange and Yoshida.

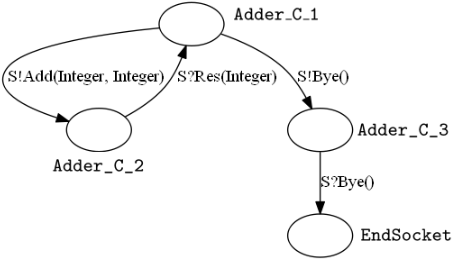
"State Channel" API generation



- Protocol states as *state-specific* channel types
  - Java nominal types: state enumeration as default
  - Generated *state channel* offers exactly the valid I/O operations for the corresponding protocol state
    - Three kinds: output, unary input, non-unary input
  - Fluent interface for chaining channel operations through successive states
    - Only the initial state channel class has a public constructor

[FASE16] *Hybrid Session Verification through Endpoint API Generation.* Hu and Yoshida.

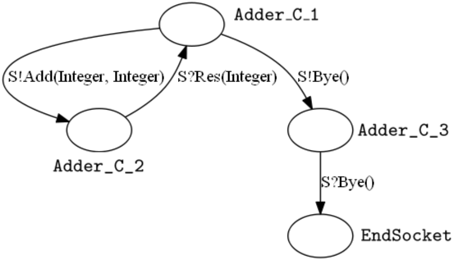
"State Channel" API generation



- Protocol states as *state-specific* channel types
  - Java nominal types: state enumeration as default
  - Generated *state channel* offers exactly the valid I/O operations for the corresponding protocol state
    - Three kinds: output, unary input, non-unary input
  - Fluent interface for chaining channel operations through successive states
    - Only the initial state channel class has a public constructor

[FASE16] *Hybrid Session Verification through Endpoint API Generation.* Hu and Yoshida.

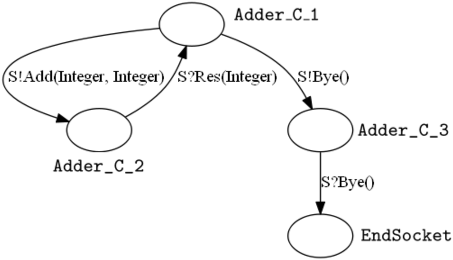
“State Channel” API generation



- Protocol states as *state-specific* channel types
  - Java nominal types: state enumeration as default
  - Generated *state channel* offers exactly the valid I/O operations for the corresponding protocol state
    - Three kinds: output, unary input, non-unary input
  - Fluent interface for chaining channel operations through successive states
    - Only the initial state channel class has a public constructor

[FASE16] *Hybrid Session Verification through Endpoint API Generation.* Hu and Yoshida.

“State Channel” API generation



- Protocol states as *state-specific* channel types
  - Java nominal types: state enumeration as default
  - Generated *state channel* offers exactly the valid I/O operations for the corresponding protocol state
    - Three kinds: output, unary input, non-unary input
  - Fluent interface for chaining channel operations through successive states
    - Only the initial state channel class has a public constructor

[FASE16] *Hybrid Session Verification through Endpoint API Generation.* Hu and Yoshida.

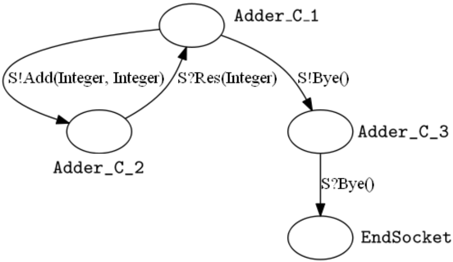
Example: Adder

- Reify session type names as Java singleton types
- Main “Session” class

```
public final class Adder extends Session {
  public static final C C = C.C;
  public static final S S = S.S;
  public static final Add Add = Add.Add;
  public static final Bye Bye = Bye.Bye;
  public static final Res Res = Res.Res;
  ...
```

  - Instances represent run-time sessions of this (initial) type in execution
    - Encapsulates source protocol info, run-time session ID, etc.

Adder: State Channel API for C



- `Adder_C_1`
  - Output state channel: (overloaded) `send` methods

```
Adder_C_2 send(S role, Add op, Integer arg0, Integer arg1) throws ...
Adder_C_3 send(S role, Bye op) throws ...
```

    - Parameter types: message recipient, operator and payload
    - Return type: successor state

Adder: State Channel API for C

Class Adder_C_1

java.lang.Object
    org.scribble.net.scribsock.ScribSocket<S,R>
        org.scribble.net.scribsock.LinearSocket<S,R>
            org.scribble.net.scribsock.SendSocket<Adder,C>
                demo.fase.adder.Adder.Adder.channels.C.Adder_C_1

All Implemented Interfaces:

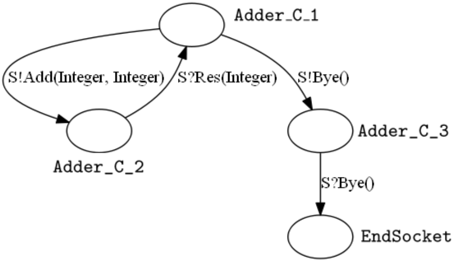
Out_S_Add_Integer_Integer<Adder_C_2>, Out_S_Bye<Adder_C_3>, Select_C_S_Add_Integer_Integer<Adder_C_2>, Select_C_S_Add_Integer_Integer__S_Bye<Adder_C_2,Adder_C_3>, Select_C_S_Bye<Adder_C_3>, Succ_In_S_Res_Integer

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
Adder_C_2	send(S role, Add op, java.lang.Integer arg0, java.lang.Integer arg1)
Adder_C_3	send(S role, Bye op)

Adder: State Channel API for C



- `Adder_C_2`

  ```
  Adder_C_1 receive(S role, Res op, Buf<? super Integer> arg1) throws ...
  ```

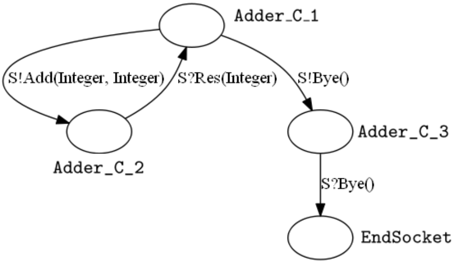
  - Unary input state channel: a `receive` method
  - (Received payload written to a parameterised buffer arg)
  - Recursion: return new instance of a "previous" channel type

- `Adder_C_3`

  ```
  EndSocket receive(S role, Bye op) throws ...
  ```

  - `EndSocket` for terminal state

Adder: endpoint implementation for C

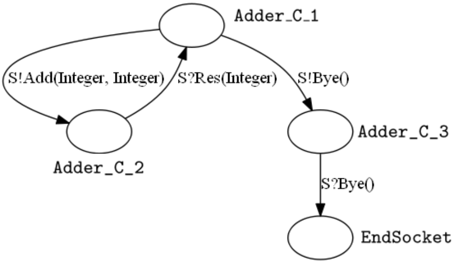



```
Adder_C_1 c1 = new Adder_C_1(...);
Buf<Integer> i = new Buf<>(1);
while (i.val < N)
  c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
c1.send(S, Bye).receive(S, Bye);
```

EndSocket Adder_C_3.receive(S role, Bye op) throws ScribbleRuntimeException, IOException, ClassNotFoundException

- Implicit API usage contract:
  - Use each state channel instance exactly once
    - "Hybrid" enforcement of session types:
      Linear usage of channel instances checked at run-time by generated API

Adder: endpoint implementation for C




```
Adder_C_1 c1 = new Adder_C_1(...);
Buf<Integer> i = new Buf<>(1);
while (i.val < N)
  c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
c1.send(S, Bye).receive(S, Bye);
```

- Implicit API usage contract:
  - Use each state channel instance exactly once
    - "Hybrid" enforcement of session types:
      Linear usage of channel instances checked at run-time by generated API

“Hybrid” enforcement of session types

```
Adder adder = new Adder();
try (SessionEndpoint<Adder, C> client
     = new SessionEndpoint<>(adder, C, ...)) {
  client.connect(S, SocketChannelEndpoint::new, host, port);
  Adder_C_1 c1 = new Adder_C_1(client);
  Buf<Integer> i = new Buf<>(1);
  while (i.val < N)
    c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
  c1.send(S, Bye).receive(S, Bye);
}
```

- Static typing: session I/O actions as State Channel API methods
- Run-time checks: *linear* usage of state channel instances

“Hybrid” enforcement of session types

```
Adder adder = new Adder();
try (SessionEndpoint<Adder, C> client
      = new SessionEndpoint<>(adder, C, ...)) {
  client.connect(S, SocketChannelEndpoint::new, host, port);
  Adder_C_1 c1 = new Adder_C_1(client);
  Buf<Integer> i = new Buf<>(1);
  while (i.val < N)
    c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
  c1.send(S, Bye).receive(S, Bye);
}
```

- Static typing: session I/O actions as State Channel API methods
- Run-time checks: *linear* usage of state channel instances
  - At most once
    - “Used” flag per *channel* instance checked and set by I/O actions
  - At least once
    - “Complete” flag per *endpoint* instance set by *terminal action*
    - Checked via `try` on `AutoCloseable` `SessionEndpoint`

“Hybrid” enforcement of session types

```
Adder adder = new Adder();
try (SessionEndpoint<Adder, C> client
      = new SessionEndpoint<>(adder, C, ...)) {
  client.connect(S, SocketChannelEndpoint::new, host, port);
  Adder_C_1 c1 = new Adder_C_1(client);
  Buf<Integer> i = new Buf<>(1);
  while (i.val < N)
    c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
  c1.send(S, Bye).receive(S, Bye);
}
```

- Static typing: session I/O actions as State Channel API methods
- Run-time checks: *linear* usage of state channel instances
  - At most once
    - “Used” flag per *channel* instance checked and set by I/O actions
  - At least once
    - “Complete” flag per *endpoint* instance set by *terminal action*
    - Checked via `try` on `AutoCloseable` `SessionEndpoint`

“Hybrid” enforcement of session types

```
Adder adder = new Adder();
try (SessionEndpoint<Adder, C> client
      = new SessionEndpoint<>(adder, C, ...)) {
  client.connect(S, SocketChannelEndpoint::new, host, port);
  Adder_C_1 c1 = new Adder_C_1(client);
  Buf<Integer> i = new Buf<>(1);
  while (i.val < N)
    c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
  c1.send(S, Bye).receive(S, Bye);
}
```

- Static typing: session I/O actions as State Channel API methods
- Run-time checks: *linear* usage of state channel instances
  - At most once
    - “Used” flag per *channel* instance checked and set by I/O actions
  - At least once
    - “Complete” flag per *endpoint* instance set by *terminal action*
    - Checked via `try` on `AutoCloseable` `SessionEndpoint`

"Hybrid" enforcement of session types

```
Adder adder = new Adder();
try (SessionEndpoint<Adder, C> client
      = new SessionEndpoint<>(adder, C, ...)) {
  client.connect(S, SocketChannelEndpoint::new, host, port);
  Adder_C_1 c1 = new Adder_C_1(client);
  Buf<Integer> i = new Buf<>(1);
  while (i.val < N)
    c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
  c1.send(S, Bye).receive(S, Bye);
}
```

- Static typing: session I/O actions as State Channel API methods
- Run-time checks: *linear* usage of state channel instances
  - At most once
    - "Used" flag per *channel* instance checked and set by I/O actions
  - At least once
    - "Complete" flag per *endpoint* instance set by *terminal action*
    - Checked via `try` on `AutoCloseable` `SessionEndpoint`

“Hybrid” enforcement of session types

```
Adder adder = new Adder();
try (SessionEndpoint<Adder, C> client
      = new SessionEndpoint<>(adder, C, ...)) {
  client.connect(S, SocketChannelEndpoint::new, host, port);
  Adder_C_1 c1 = new Adder_C_1(client);
  Buf<Integer> i = new Buf<>(1);
  while (i.val < N)
    c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
  c1.send(S, Bye).receive(S, Bye);
}
```

- Static typing: session I/O actions as State Channel API methods
- Run-time checks: *linear* usage of state channel instances
  - At most once
    - “Used” flag per *channel* instance checked and set by I/O actions
  - At least once
    - “Complete” flag per *endpoint* instance set by *terminal action*
    - Checked via `try` on `AutoCloseable` `SessionEndpoint`

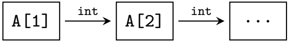
“Hybrid” enforcement of session types

```
Adder adder = new Adder();
try (SessionEndpoint<Adder, C> client
      = new SessionEndpoint<>(adder, C, ...)) {
  client.connect(S, SocketChannelEndpoint::new, host, port);
  Adder_C_1 c1 = new Adder_C_1(client);
  Buf<Integer> i = new Buf<>(1);
  while (i.val < N)
    c1 = c1.send(S, Add, i.val, i.val).receive(S, Res, i);
  c1.send(S, Bye).receive(S, Bye);
}
```

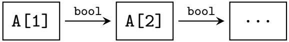
- Static typing: session I/O actions as State Channel API methods
- Run-time checks: *linear* usage of state channel instances
- API generation safety properties:
  - If state channel linearity respected:
    Communication safety (e.g. [JACM16] Error-freedom) satisfied
  - Regardless of linearity: non-compliant I/O actions *never* executed
    (up to premature termination)

Extensions: parameterised role structures

```
global protocol Ring(role W(k)) {
  choice at W[1] {
    Next  dot  W[1..k-1] to W[2..k];
    Res   from W[k]      to W[1];
  } or {
    Done  dot W[1..k-1] to W[2..k];
  }
}
```



or



$$\mathtt{W}[1..k-1] \xrightarrow{1} \mathtt{W}[2..k] : \{\mathtt{Next}\,.\,\widetilde{G}_1, \mathtt{Done}\}$$

- Challenge: distribution – i.e., well-formedness and projection
- Target language: Go
  - “Heterogeneous” communications: first class shared mem. channels and traditional distributed programming libraries

$$\widetilde{G} \restriction \mathtt{W}\{1, 1..k{-}1\} = \mathbf{rec}\, X\, \mathbf{!}\langle \mathtt{W}[\mathbf{id}{+}1], \{\mathtt{Res}\,.\,\mathbf{?}\langle \mathtt{W}[k], \mathtt{Res}\rangle\,;\, X\,,\, \mathtt{done}\,.\,\mathbf{0}\}\rangle$$

$$\widetilde{G} \restriction \mathtt{W}\{1..k{-}1, 2..k\} = \mathbf{rec}\, X\, \mathbf{?}\langle \mathtt{W}[\mathbf{id}{-}1], \{\mathtt{Res}\,.\,\mathbf{!}\langle \mathtt{W}[\mathbf{id}{+}1], \mathtt{Res}\rangle\,;\, X\,,\, \mathtt{done}\,.\,\mathbf{!}\langle \mathtt{W}[\mathbf{id}{+}1], \mathtt{done}\rangle\}\rangle$$

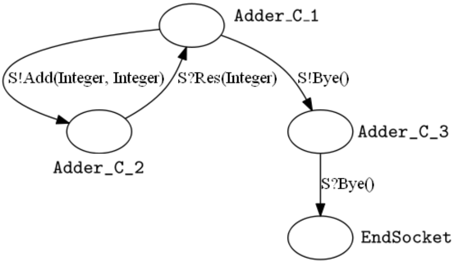
$$\widetilde{G} \restriction \mathtt{W}\{2..k, 2\} = \mathbf{rec}\, X\, \mathbf{?}\langle \mathtt{W}[\mathbf{id}{-}1], \{\mathtt{Res}\,.\,\mathbf{!}\langle \mathtt{W}[1], \mathtt{Res}\rangle\,;\, X\,,\, \mathtt{done}\,.\,\mathbf{0}\}\rangle$$

Extensions: “interaction refinements”

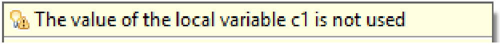
```
global protocol MyProto(role A, role B, role C) {
  1(x: int) from A to B; @x>0
  2(x) from B to C;
  choice at C {
    3() from C to A; @x>5
  } or {
    4() from C to A; @x<=5
} }
```

- Aims: message type refinements, assertions on I/O action and session state, message value dependent control flow, ...
- Target language: F# (.NET) – type providers
- Approach: static verification by integrating Scribble with SMT solver, correctness-by-construction from code generation, and automated assertion inlining

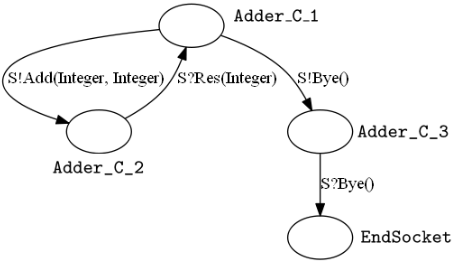
Adder: endpoint implementation for C



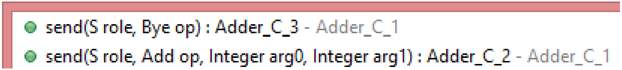

```
Adder_C_1 c1 = new Adder_C_1(...);
```



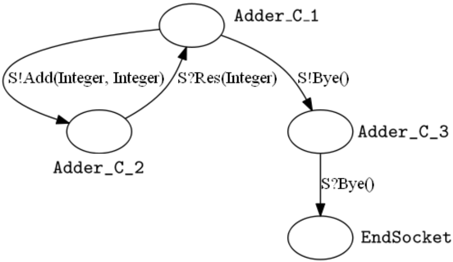
Adder: endpoint implementation for C




```
Adder_C_1 c1 = new Adder_C_1(...);

c1.
```



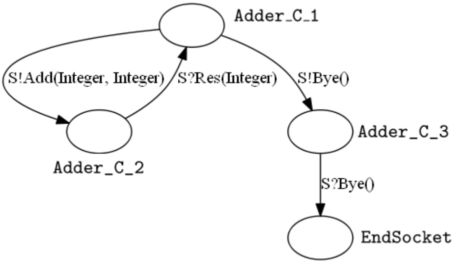
Adder: endpoint implementation for C



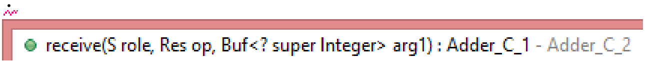

```
Adder_C_1 c1 = new Adder_C_1(...);
Buf<Integer> i = new Buf<>(1);
c1.send(S, Add, i.val, i.val);
```

Adder_C_2 Adder_C_1.send(S role, Add op, Integer arg0, Integer arg1) throws ScribbleRuntimeException, IOException

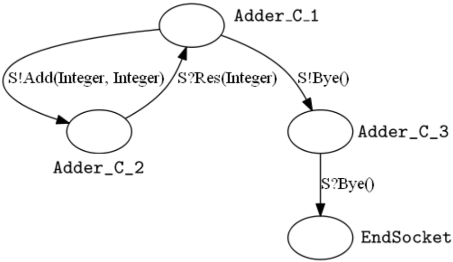
Adder: endpoint implementation for C



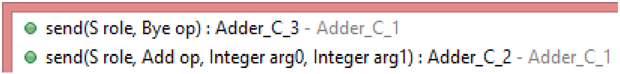

```
Adder_C_1 c1 = new Adder_C_1(...);
Buf<Integer> i = new Buf<>(1);
c1.send(S, Add, i.val, i.val)
```



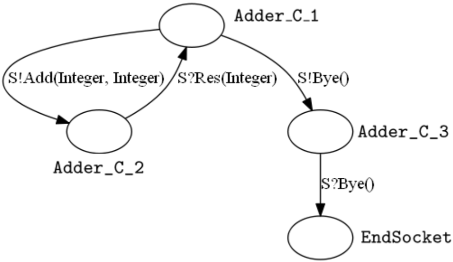
Adder: endpoint implementation for C




```
Adder_C_1 c1 = new Adder_C_1(...);
Buf<Integer> i = new Buf<>(1);
c1.send(S, Add, i.val, i.val)
  .receive(S,Res, i)
  .send(S, Add, i.val, i.val)
  .receive(S,Res, i)
  .send(S, Add, i.val, i.val)
  .receive(S,Res, i)
  .
```



Adder: endpoint implementation for C




```
Adder_C_1 c1 = new Adder_C_1(...);
Buf<Integer> i = new Buf<>(1);
c1.send(S, Add, i.val, i.val)
  .receive(S,Res, i)
  .send(S, Add, i.val, i.val)
  .receive(S,Res, i)
  //.send(S, Add, i.val, i.val)
  .receive(S,Res, i)
```

The method receive(S, Res, Buf<Integer>) is undefined for the type Adder_C_1