Explicit Connection Actions in Multiparty Session Types

Raymond Hu and Nobuko Yoshida

Imperial College London

Outline

- Background: multiparty session types (MPST)
  - Scribble: ongoing work on implementing and applying MPST to practice
  - × Standard MPST do not support sessions with *dynamic or optional involvement of participants*

- MPST with explicit connection actions
  - MP sessions as a dynamically evolving configuration of binary connections
    - Modelling-based well-formedness for MPST protocols
    - Session subtyping and role progress
    - Multiparty correlation of binary connections
  - Motivating examples
    - Web services choreography (Travel Agency)
    - Microservices industry use case (Supplier Info)
    - Standardised application-layer protocol (FTP)

MPST (background)

- Standard presentation: three-layer framework
  - Global type

    $G = A \to B : \langle U_1 \rangle \,.\, B \to C : \langle U_2 \rangle \,.\, C \to A : \langle U_3 \rangle$

    Global description of multiparty *message passing* protocol/choreography

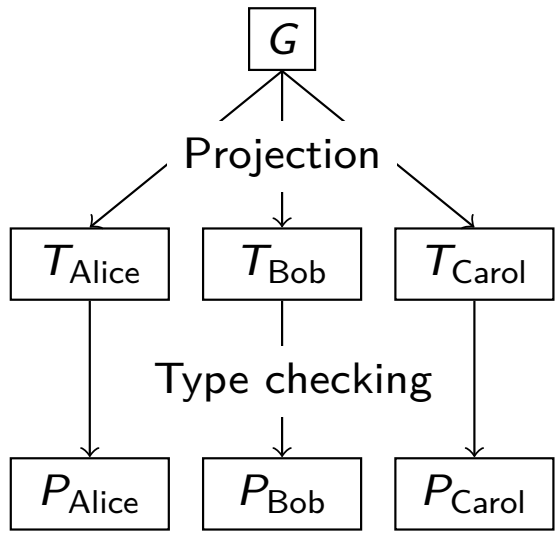
    Participants abstracted as *roles*
  - Local types

    $T_A = !\langle B, U_1 \rangle \,.\, ?\langle C, U_3 \rangle$

    Localised view of the protocol for each role
  - Endpoint processes

    $P_A = a[A](x) \,.\, x!\langle B, u_1 \rangle \,.\, x?(C, y)$

    Perform I/O via special primitives on channels



- *Communication safety* is ensured for a parallel composition of well-typed endpoints

[POPL08] *Multiparty asynchronous session types*. Honda, Yoshida and Carbone.

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida.

MPST (background)

- Standard presentation: three-layer framework
  - Global type

    $$G = A \to B : \langle U_1 \rangle \,.\, B \to C : \langle U_2 \rangle \,.\, C \to A : \langle U_3 \rangle$$

    Global description of multiparty *message passing* protocol/choreography

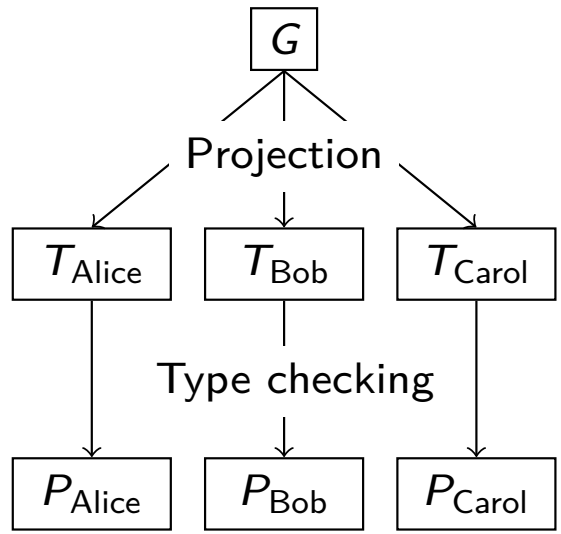
    Participants abstracted as *roles*
  - Local types

    $$T_A = !\langle B, U_1 \rangle \,.\, ?\langle C, U_3 \rangle$$

    Localised view of the protocol for each role
  - Endpoint processes

    $$P_A = a[A](x) \,.\, x!\langle B, u_1 \rangle \,.\, x?(C, y)$$

    Perform I/O via special primitives on channels



- *Communication safety* is ensured for a parallel composition of well-typed endpoints

[POPL08] *Multiparty asynchronous session types*. Honda, Yoshida and Carbone.

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida.

MPST (background)

- Standard presentation: three-layer framework
  - Global type

    $$G = A \to B : \langle U_1 \rangle \,.\, B \to C : \langle U_2 \rangle \,.\, C \to A : \langle U_3 \rangle$$

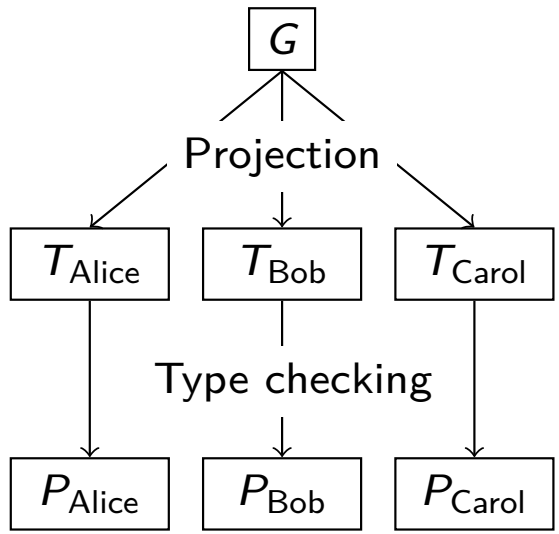
    Global description of multiparty *message passing* protocol/choreography
    Participants abstracted as *roles*
  - Local types

    $$T_A = !\langle B, U_1 \rangle \,.\, ?\langle C, U_3 \rangle$$

    Localised view of the protocol for each role
  - Endpoint processes

    $$P_A = a[A](x) \,.\, x!\langle B, u_1 \rangle \,.\, x?(C, y)$$

    Perform I/O via special primitives on channels



- *Communication safety* is ensured for a parallel composition of well-typed endpoints

[POPL08] *Multiparty asynchronous session types*. Honda, Yoshida and Carbone.

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida.

MPST (background)

- Standard presentation: three-layer framework
  - Global type

    $$G = A \to B : \langle U_1 \rangle \,.\, B \to C : \langle U_2 \rangle \,.\, C \to A : \langle U_3 \rangle$$

    Global description of multiparty *message passing* protocol/choreography

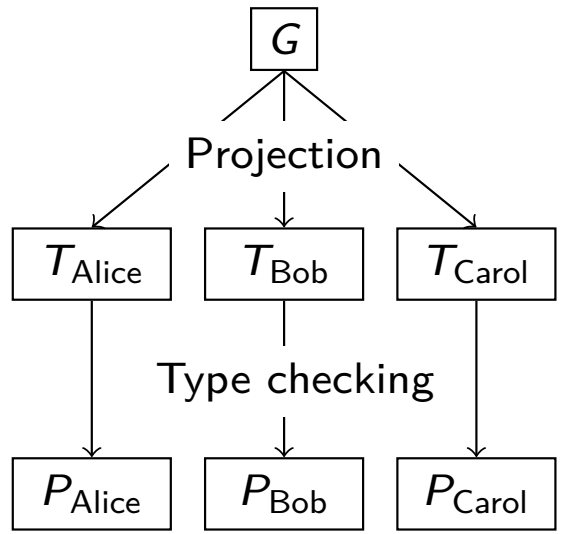
    Participants abstracted as *roles*
  - Local types

    $$T_A = !\langle B, U_1 \rangle \,.\, ?\langle C, U_3 \rangle$$

    Localised view of the protocol for each role
  - Endpoint processes

    $$P_A = a[A](x) \,.\, x!\langle B, u_1 \rangle \,.\, x?(C, y)$$

    Perform I/O via special primitives on channels



- *Communication safety* is ensured for a parallel composition of well-typed endpoints

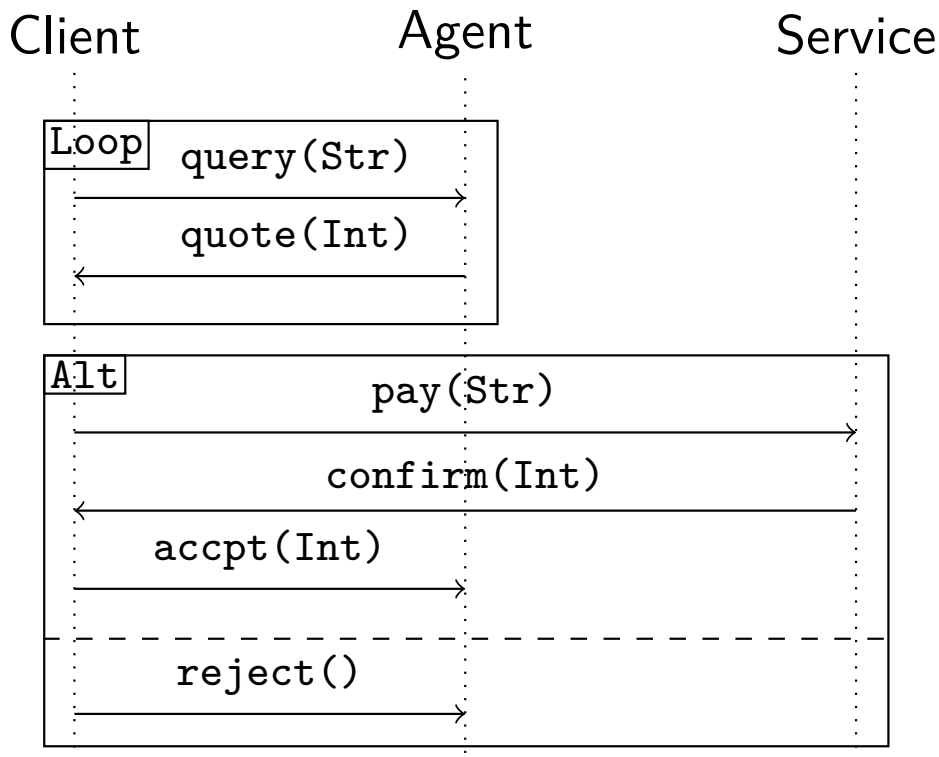
[POPL08] *Multiparty asynchronous session types*. Honda, Yoshida and Carbone.

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida.

Web Services use case: Travel Agency

- W3C Choreography working group requirements use case

  `https://www.w3.org/TR/ws-chor-reqs/#UC-001`



[ECOOP06] *Session Types for Object-Oriented Languages*. Dezani-Ciancaglini, Mostrous, Yoshida and Drossopoulou. "Buyer-Seller-Shipper"

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida. "Three-Buyer"

[FTPL16] *Behavioral Types in Programming Languages*. Ancona et al. "Customer-Agency"

Travel Agency in MPST (naive attempt)

- As a Scribble global protocol (asynchronous MPST global type):

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```

× Not a valid global type

- "Unfinished role" error
- Ruled out by *syntactic* well-formedness:
  Each involved participant must be present in *all* choice cases

- A session cannot have dynamic or optional involvement of participants
  - MPST literature uses work arounds: *e.g.*, adding extra messages, decomposing into separate protocols, session *delegation*, ...

Travel Agency in MPST (naive attempt)

- As a Scribble global protocol (asynchronous MPST global type):

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```

× Not a valid global type

  - "Unfinished role" error
  - Ruled out by *syntactic* well-formedness:
    Each involved participant must be present in *all* choice cases

- A session cannot have dynamic or optional involvement of participants
  - MPST literature uses work arounds: *e.g.*, adding extra messages, decomposing into separate protocols, session *delegation*, ...

Travel Agency in MPST (naive attempt)

- As a Scribble global protocol (asynchronous MPST global type):

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```

× Not a valid global type

  - "Unfinished role" error
  - Ruled out by *syntactic* well-formedness:
    Each involved participant must be present in *all* choice cases

- A session cannot have dynamic or optional involvement of participants
  - MPST literature uses work arounds: *e.g.*, adding extra messages, decomposing into separate protocols, session *delegation*, ...

Travel Agency in MPST (naive attempt)

- As a Scribble global protocol (asynchronous MPST global type):

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```

- × Not a valid global type
  - "Unfinished role" error
  - Ruled out by *syntactic* well-formedness:
    Each involved participant must be present in *all* choice cases
- A session cannot have dynamic or optional involvement of participants
  - MPST literature uses work arounds: *e.g.*, adding extra messages, decomposing into separate protocols, session *delegation*, ...

Travel Agency in MPST (naive attempt)

- As a Scribble global protocol (asynchronous MPST global type):

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```

× Not a valid global type

  - "Unfinished role" error
  - Ruled out by *syntactic* well-formedness:
    Each involved participant must be present in *all* choice cases

- A session cannot have dynamic or optional involvement of participants
  - MPST literature uses work arounds: *e.g.*, adding extra messages, decomposing into separate protocols, session *delegation*, ...

Travel Agency in MPST (naive attempt)

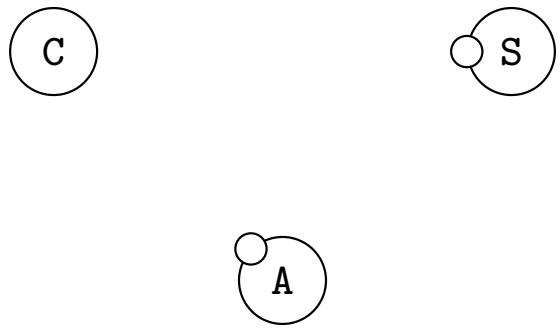
- As a Scribble global protocol (asynchronous MPST global type):

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```

- × Not a valid global type
  - "Unfinished role" error
  - Ruled out by *syntactic* well-formedness:
    Each involved participant must be present in *all* choice cases
- A session cannot have dynamic or optional involvement of participants
  - MPST literature uses work arounds: *e.g.*, adding extra messages, decomposing into separate protocols, session *delegation*, ...

Standard MPST operational semantics

- Session initation by a global atomic synchronisation
- Sender-asynchronous (non-blocking output), reliable, role-to-role ordering

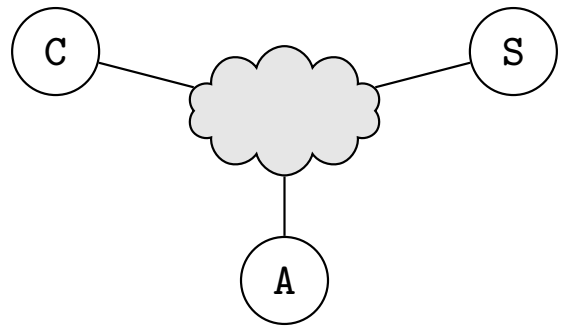


- MPST safety: run-time session execution is safe from
  - Reception errors
  - Deadlocks
  - Orphan messages

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida.

Standard MPST operational semantics

- Session initation by a global atomic synchronisation
- Sender-asynchronous (non-blocking output), reliable, role-to-role ordering

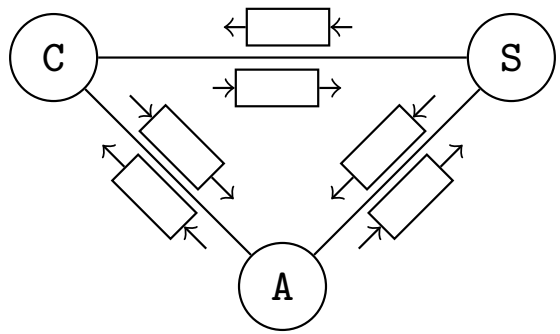


- MPST safety: run-time session execution is safe from
  - Reception errors
  - Deadlocks
  - Orphan messages

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida.

Standard MPST operational semantics

- Session initation by a global atomic synchronisation
- Sender-asynchronous (non-blocking output), reliable, role-to-role ordering

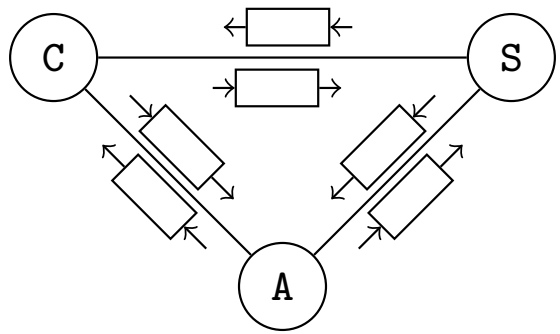


- MPST safety: run-time session execution is safe from
  - Reception errors
  - Deadlocks
  - Orphan messages

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida.

Standard MPST operational semantics

- Session initation by a global atomic synchronisation
- Sender-asynchronous (non-blocking output), reliable, role-to-role ordering



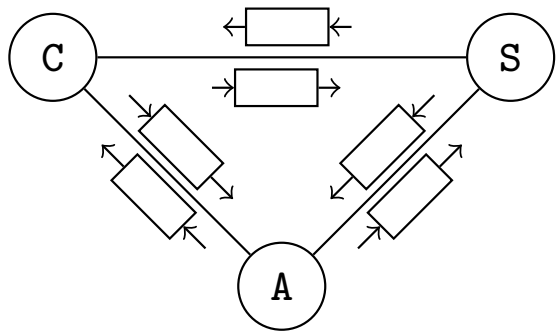
- MPST safety: run-time session execution is safe from
  - Reception errors
  - Deadlocks
  - Orphan messages

[CONCUR08] *Global progress in dynamically interleaved multiparty sessions*. Bettini, Coppo, D'Antoni, Luca, Dezani-Ciancaglini and Yoshida.

Travel Agency in MPST (naive attempt)

- As a Scribble global protocol (asynchronous MPST global type):

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```

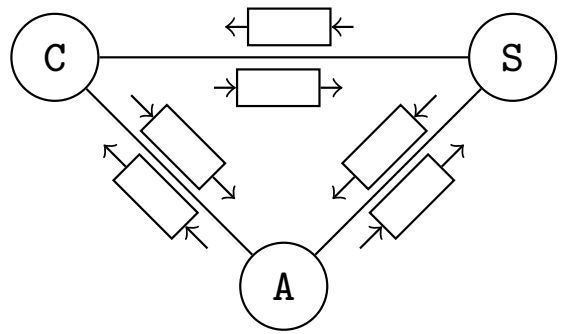


- × Not a valid global type
  - "Unfinished role" error
  - Ruled out by *syntactic* well-formedness:
    Each involved participant must be present in *all* choice cases
- A session cannot have dynamic or optional involvement of participants
  - MPST literature uses work arounds: *e.g.*, adding extra messages, decomposing into separate protocols, session *delegation*, ...

Travel Agency in MPST (naive attempt)

- As a Scribble global protocol (asynchronous MPST global type):

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



× Not a valid global type

  - "Unfinished role" error
  - Ruled out by *syntactic* well-formedness:
    Each involved participant must be present in *all* choice cases

- A session cannot have dynamic or optional involvement of participants
  - MPST literature uses work arounds: *e.g.*, adding extra messages, decomposing into separate protocols, session *delegation*, ...

Outline

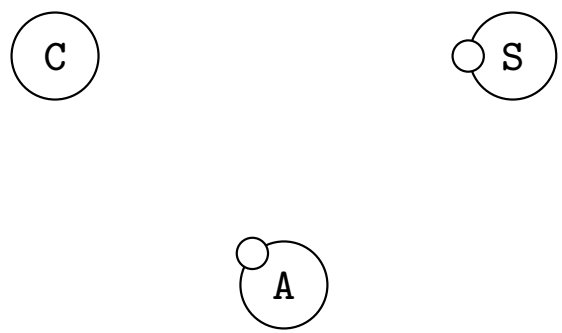
- Background: multiparty session types (MPST)
  - Scribble: ongoing work on implementing and applying MPST to practice
  - × Standard MPST do not support sessions with *dynamic or optional involvement of participants*

- MPST with explicit connection actions
  - MP sessions as a dynamically evolving configuration of binary connections
    - Modelling-based well-formedness for MPST protocols
    - Session subtyping and role progress
    - Multiparty correlation of binary connections
  - Motivating examples
    - Web services choreography (Travel Agency)
    - Microservices industry use case (Supplier Info)
    - Standardised application-layer protocol (FTP)

Travel Agency in MPST with explicit connections

- Practical protocol specifications include explicit connection request/accept (and disconnect) actions

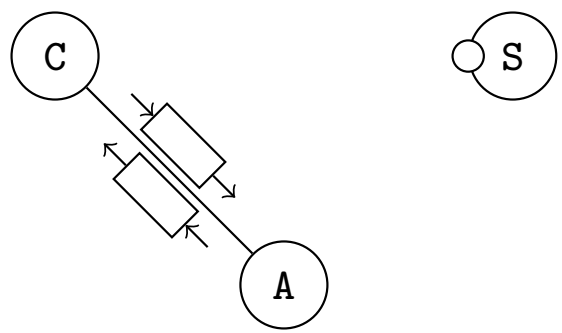
```
explicit global protocol Travel(role C, role A, role S) {
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Travel Agency in MPST with explicit connections

- Practical protocol specifications include explicit connection request/accept (and disconnect) actions

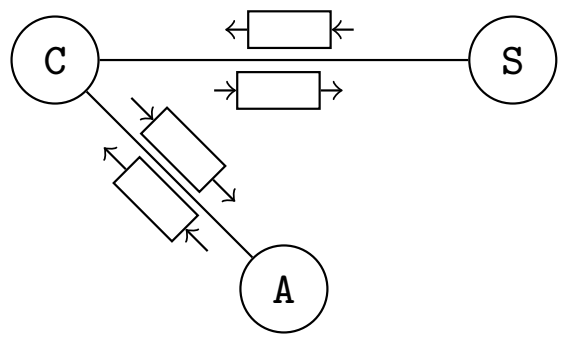
```
explicit global protocol Travel(role C, role A, role S) {
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Travel Agency in MPST with explicit connections

- Practical protocol specifications include explicit connection request/accept (and disconnect) actions

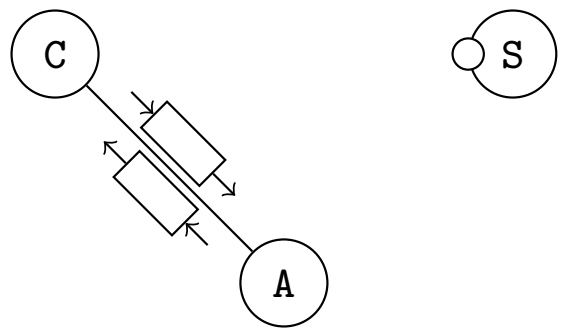
```
explicit global protocol Travel(role C, role A, role S) {
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Travel Agency in MPST with explicit connections

- Practical protocol specifications include explicit connection request/accept (and disconnect) actions

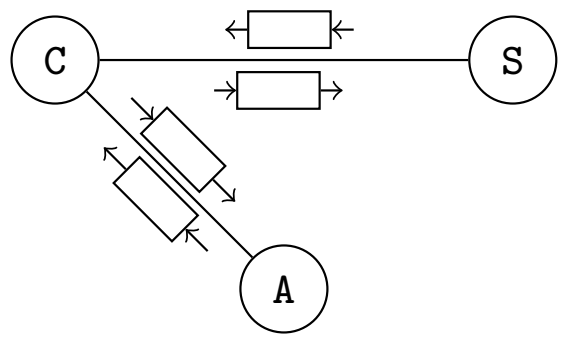
```
explicit global protocol Travel(role C, role A, role S) {
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Travel Agency in MPST with explicit connections

- Practical protocol specifications include explicit connection request/accept (and disconnect) actions

```
explicit global protocol Travel(role C, role A, role S) {
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Validating MPST with explicit connection actions

- Dynamically established binary connections
  - ▸ Role involvement guarded by initial connection accept
  - × Cannot apply standard (conservative) syntactic MPST well-formedness

- Previous works have studied MPST safety in terms of CFSM-based well-formedness conditions (*multiparty compatibility*)

  [ICALP13] *Multiparty Compatibility in Communicating Automata*. Deniélou and Yoshida.
  [CONCUR15] *Meeting Deadlines Together*. Bocchi, Lange and Yoshida.

- Our approach: MPST protocol validation by a combination of syntactic constraints and explicit error checking
  - Adapt basic MPST syntactic constraints to our extended setting...
  - ...that ensure soundness of checking a *1-bounded* model of the protocol

Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

- Global type grammar
- Role enabling
- Consistent external choices

```
choice at A {
    1() from A to B;
    1() connect A to C;
    ...
} or {
    2() from A to B;
    2() connect A to C;
    ...
}
```

  - Globally-paired interactions
  - Deterministic choices

Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

- Global type grammar
- Role enabling
- Consistent external choices

```
choice at A {
    1() from A to B;
    1() connect A to C;
    ...
} or {
    2() from A to B;
    2() connect A to C;
    ...
}
```

Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

- Global type grammar
- Role enabling
- Consistent external choices

```
choice at A {
    1() from A to B;
    //1() from A to C;
    1() connect C to A;      ✗
} or {
    2() from A to B;
    2() connect A to C;
    ...
}
```

Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

- Global type grammar
- Role enabling
- Consistent external choices

```
choice at A {
    1() from A to B;
    1() connect A to C;
    ...
} or {
    2() connect A to C;
    2() from A to B;
    ...
}
```

Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

- Global type grammar
- Role enabling
- Consistent external choices

```
choice at A {
    1() from A to B;
    1() connect B to C;
    ...
} or {
    2() from A to B;
    2() connect B to C;
    ...
}
```

Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

- Global type grammar
- Role enabling
- Consistent external choices

```
choice at A {
    1() from A to B;
    1() connect A to C;
    ...
} or {
    2() from A to B;
    2() connect B to C;
    ...
}
```

Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

- Global type grammar
- Role enabling
- Consistent external choices

```
choice at A {
    1() from A to B;
    1() connect A to C;
    ...
} or {
    2() from A to B;
    2() from A to C;
    ...
}
```

Validating MPST with explicit connections

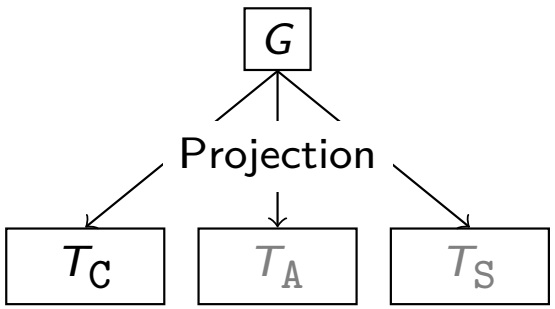
- Syntactic constraints
- MPST error checking

```
explicit global protocol Travel(role C, role A, role S) {
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```

Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

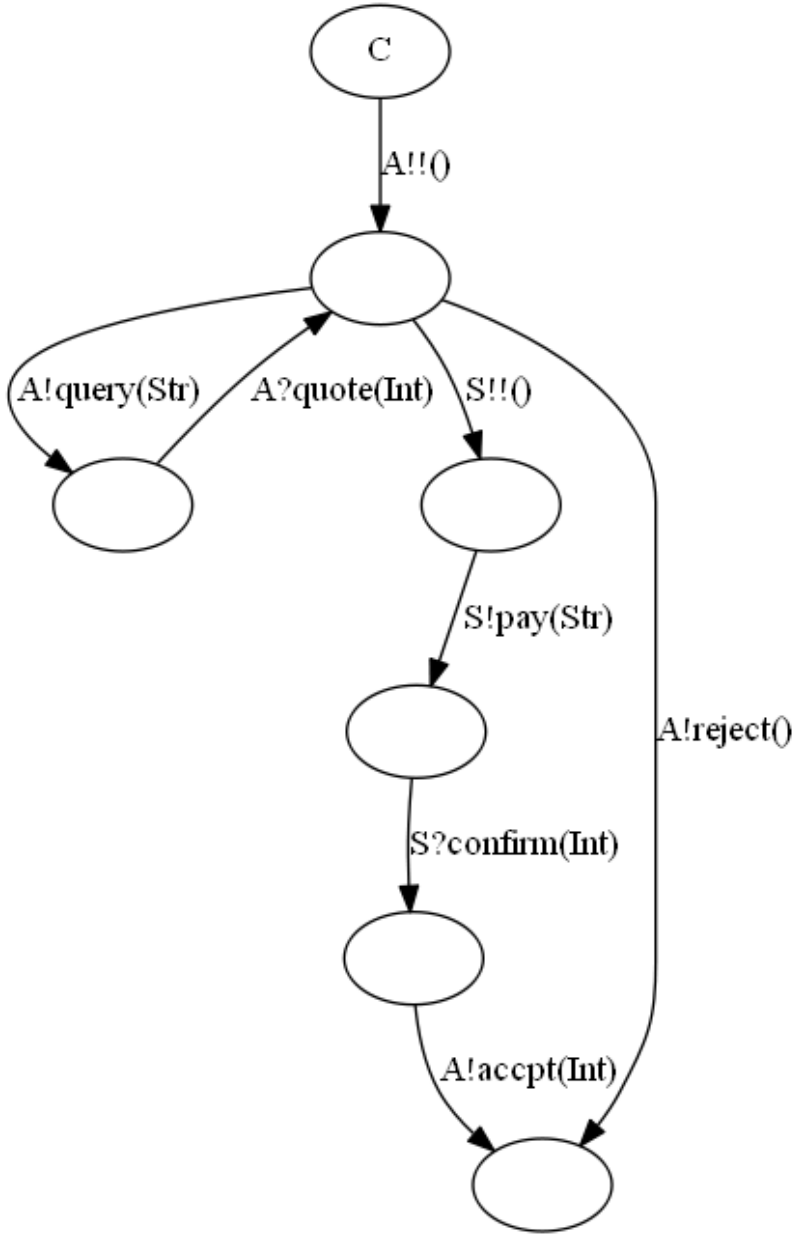
```
explicit global protocol Travel(role C, role A, role S) {
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

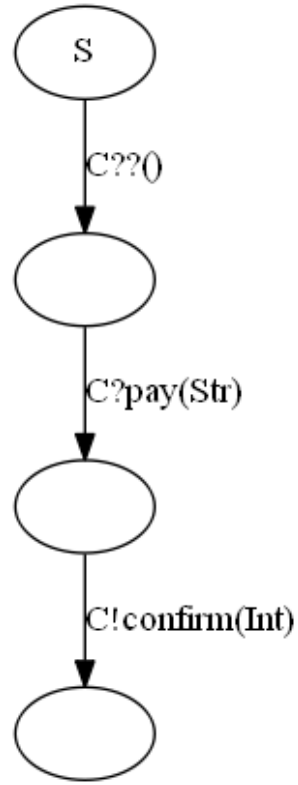
```
explicit global protocol Travel(role C, role
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, rol
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

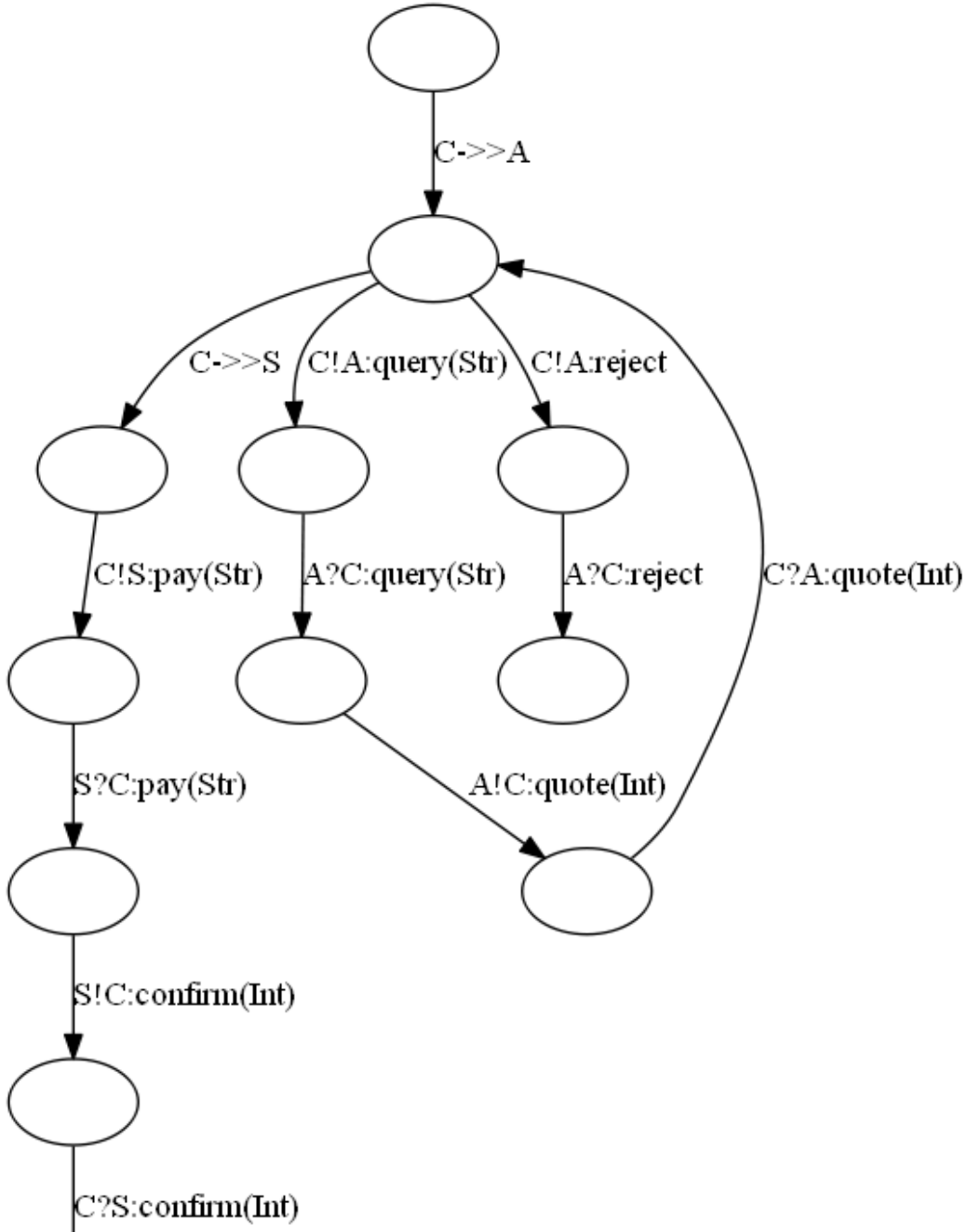
```
explicit global protocol Travel(role C, role A, role S) {
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

```
explicit global protocol Travel(ro
    connect C to A;
    do Main(C, A, S);
}

aux global protocol Main(role C, r
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Main(C, A, S);
    } or {
        connect C to S;
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject() from C to A;
}   }
```



Validating MPST with explicit connections

- Syntactic constraints
- MPST error checking

- MPST safety
  - Reception errors, Orphan messages
  - Unfinished roles, Connection/Disconnect/Unconnected errors

```
explicit global protocol Foo(role A, role B) {
    connect A to B; ... disconnect A and B; do Foo(A, B); }
```

- MPST progress
  - Eventual Reception, Role progress, Eventual Connection

- Soundness of 1-bounded MPST validation

  Let S_0 be the initial session of a wf(G) that is 1-safe and satisfies 1-progress. Then S_0 is safe and satisfies progress.

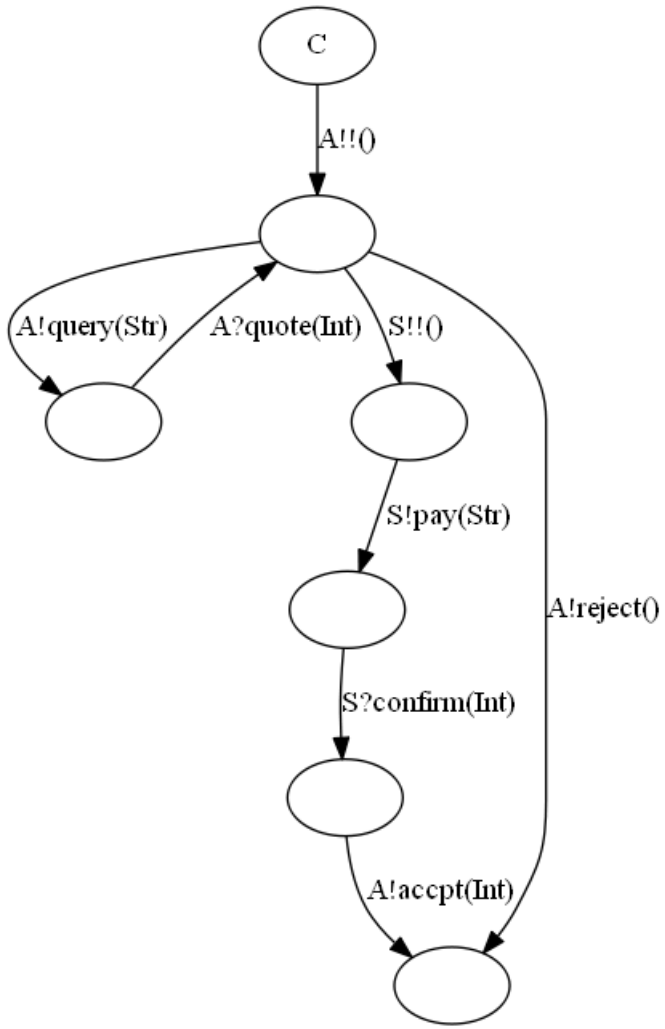
[ICALP13] *Multiparty Compatibility in Communicating Automata*. Deniélou and Yoshida.

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

MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

```
try (ExplicitEndpoint<Travel, C> ep = ...) {

        new Travel_C_1(ep)









}
```



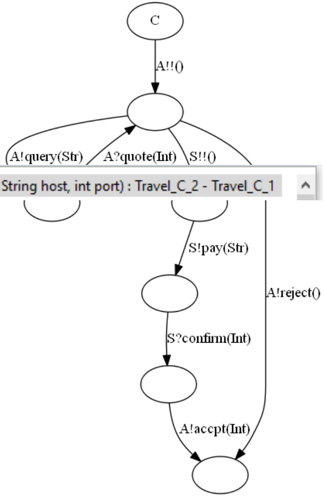
MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

```
try (ExplicitEndpoint<Travel, C> ep = ...) {

        new Travel_C_1(ep)
            .
```

connect(A role, Callable<? extends BinaryChannelEndpoint> cons, String host, int port) : Travel_C_2 - Travel_C_1

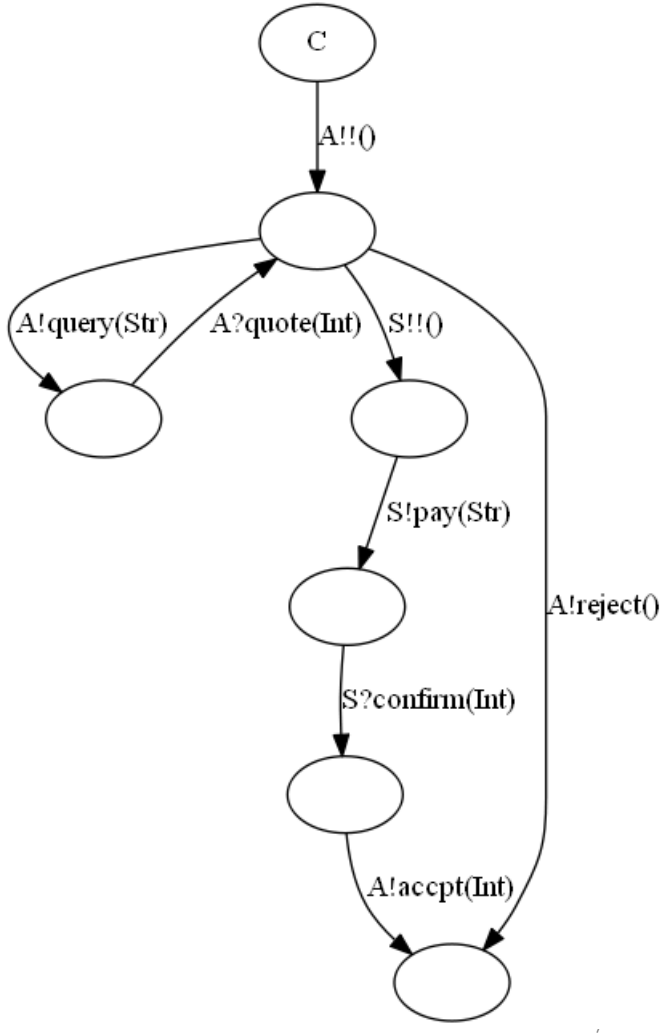
```
}
```



MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

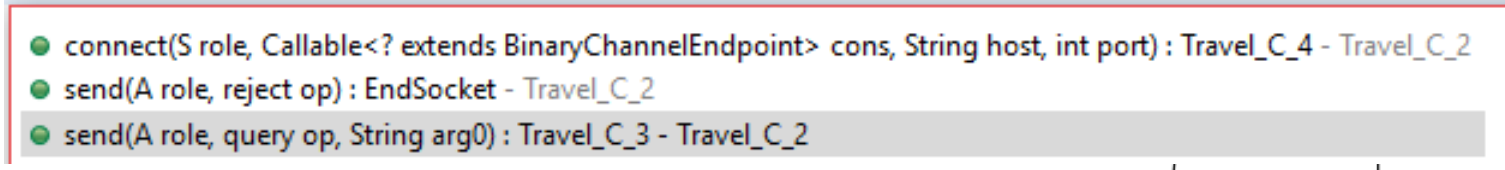
```
try (ExplicitEndpoint<Travel, C> ep = ...) {
    Travel_C_2 c2 =
        new Travel_C_1(ep)
            .connect(A, SocketChannelEndpoint::new,
                host_A, port_A);

}
```

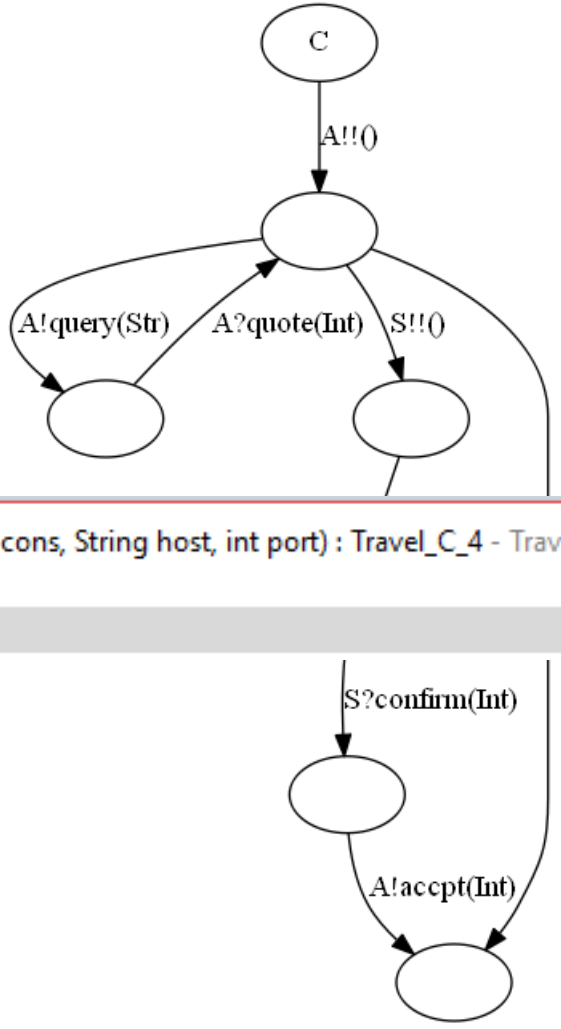


MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

```
try (ExplicitEndpoint<Travel, C> ep = ...) {
    Travel_C_2 c2 =
        new Travel_C_1(ep)
            .connect(A, SocketChannelEndpoint::new,
                host_A, port_A);
    while (doQuery())
            c2.

}
```

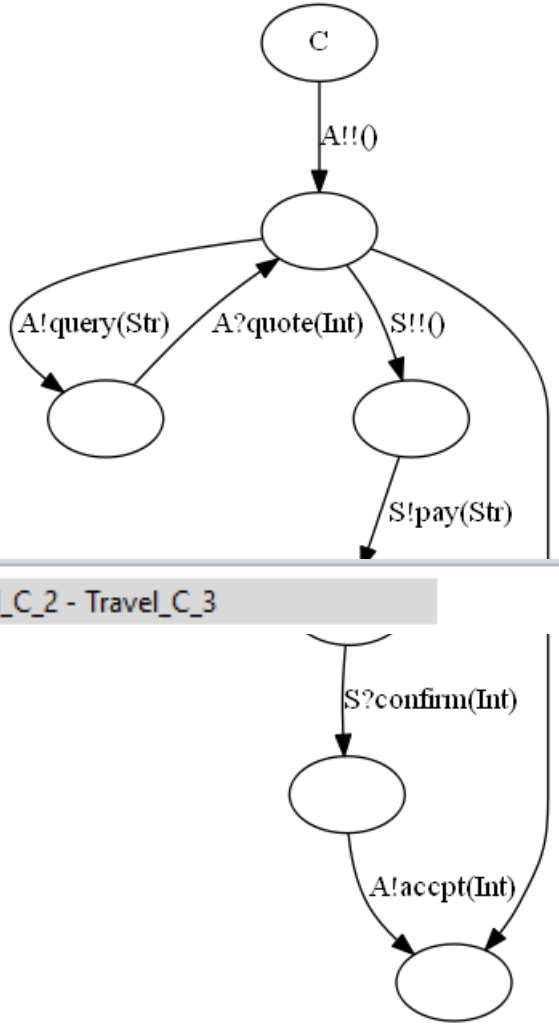




MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

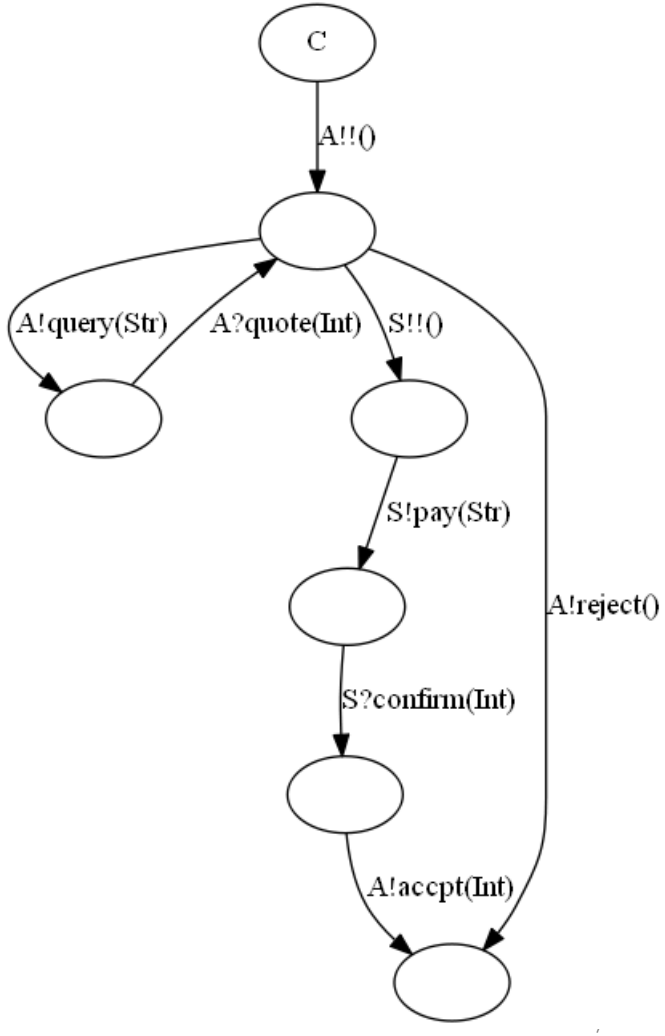
```
try (ExplicitEndpoint<Travel, C> ep = ...) {
    Travel_C_2 c2 =
        new Travel_C_1(ep)
            .connect(A, SocketChannelEndpoint::new,
                host_A, port_A);
    while (doQuery())
            c2.send(A, query, getQuery())
                .
                .
                .
}
```



MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

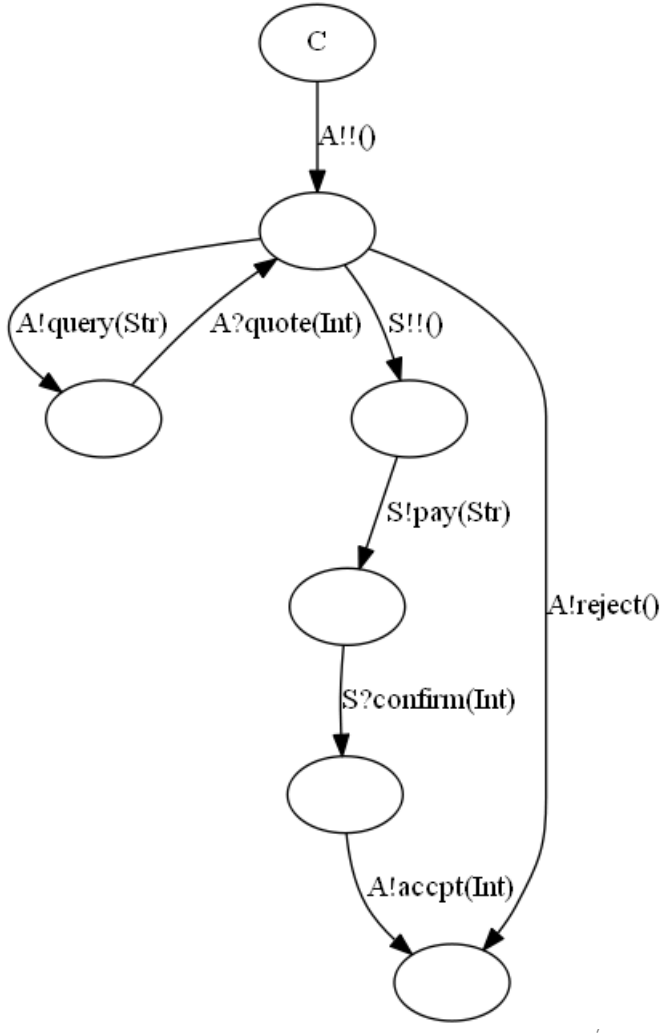
```
try (ExplicitEndpoint<Travel, C> ep = ...) {
    Travel_C_2 c2 =
        new Travel_C_1(ep)
            .connect(A, SocketChannelEndpoint::new,
                host_A, port_A);
    while (doQuery())
        c2 = c2.send(A, query, getQuery())
                .receive(A, quote, buf);



}
```



MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

```
try (ExplicitEndpoint<Travel, C> ep = ...) {
    Travel_C_2 c2 =
        new Travel_C_1(ep)
            .connect(A, SocketChannelEndpoint::new,
                host_A, port_A);
    while (doQuery())
        c2 = c2.send(A, query, getQuery())
                .receive(A, quote, buf);
    c2.connect(S, SocketChannelEndpoint::new,
            host_S, port_S)
      .send(S, pay, "payment details")
      .receive(S, confirm, buf)
      .send(A, accpt, buf.val);
}
```

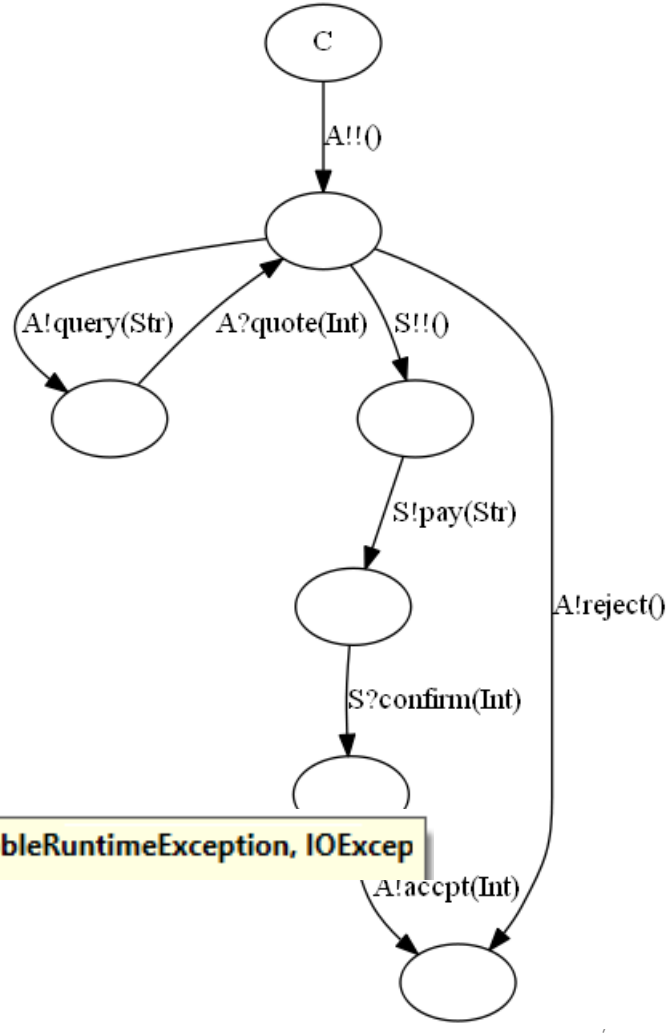


MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

```
try (ExplicitEndpoint<Travel, C> ep = ...) {
    Travel_C_2 c2 =
        new Travel_C_1(ep)
            .connect(A, SocketChannelEndpoint::new,
                host_A, port_A);
    while (doQuery())
        c2 = c2.send(A, query, getQuery())
                .receive(A, quote, buf);
    c2.connect(S, SocketChannelEndpoint::new,
            host_S, port_S)
      .send(S, pay, "payment details")
      .receive(S, confirm, buf)
      .send(A, accpt, buf.val);
}
```

EndSocket Travel_C_6.send(A role, accpt op, Integer arg0) throws ScribbleRuntimeException, IOExcep

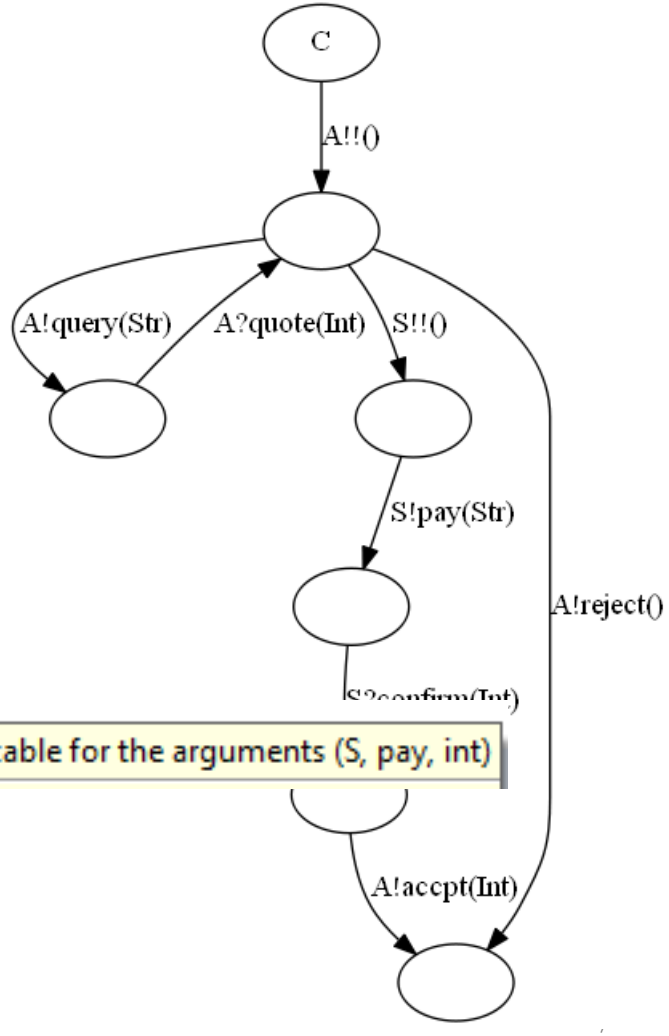


MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

```
try (ExplicitEndpoint<Travel, C> ep = ...) {
    Travel_C_2 c2 =
        new Travel_C_1(ep)
            .connect(A, SocketChannelEndpoint::new,
                host_A, port_A);
    while (doQuery())
        c2 = c2.send(A, query, getQuery())
                .receive(A, quote, buf);
    c2.connect(S, SocketChannelEndpoint::new,
            host_S, port_S)
      .send(S, pay, 1234)
      .r
      .s
}
```

The method send(S, pay, String) in the type Travel_C_4 is not applicable for the arguments (S, pay, int)

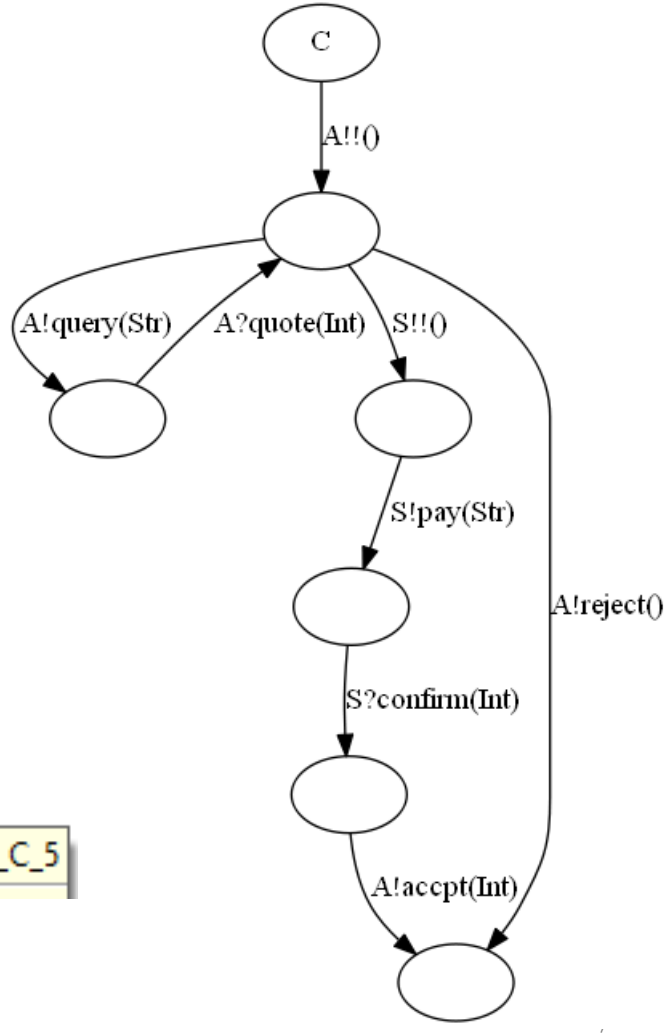


MPST endpoint implementation via API generation

- Validated MPST used to generate Java endpoint APIs
  - Generated APIs promote hybrid approach to session safety [FASE16]
    - Endpoint FSM structures captured as statically-typed call-chaining APIs
    - Usage contract of API is to use every "state channel" instance exactly once
    - Enforced by run-time channel linearity checks

```
try (ExplicitEndpoint<Travel, C> ep = ...) {
    Travel_C_2 c2 =
        new Travel_C_1(ep)
            .connect(A, SocketChannelEndpoint::new,
                host_A, port_A);
    while (doQuery())
        c2 = c2.send(A, query, getQuery())
                .receive(A, quote, buf);
    c2.connect(S, SocketChannelEndpoint::new,
            host_S, port_S)
      .send(S, pay, "payment details")
      //.receive(S, confirm, buf)
      .send(A, accpt, b.val);
}
```

The method send(A, accpt, Integer) is undefined for the type Travel_C_5



“Naive” session subtyping

- The first naive attempt at Travel Agency is invalid
  - (Non-explicit protocols checked by assuming all roles pre-connected)

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject from C to A;
}   }
```

× `S` is “unfinished” in the `reject` case

× Role progress may also be violated for `S` in the `query` case (without assuming some notion of fairness)

“Naive” session subtyping

- The first naive attempt at Travel Agency is invalid
  - (Non-explicit protocols checked by assuming all roles pre-connected)

```
global protocol Travel(role C, role A, role S) {
    choice at C {
        query(Str) from C to A;
        quote(Int) from A to C;
        do Travel(C, A, S);
    } or {
        pay(Str) from C to S;
        confirm(Int) from S to C;
        accpt(Int) from C to A;
    } or {
        reject from C to A;
}   }
```

× `S` is “unfinished” in the `reject` case

× Role progress may also be violated for `S` in the `query` case (without assuming some notion of fairness)

Output choices and role progress

- Related to *session subtyping*

```
global protocol Foo(role A, role B, role C) {
    choice at A { 1() from A to B; 1() from A to C; }
             or { 2() from A to B; 2() from A to C; }
    do Foo(A, B, C);
}
```

[ACTA05] *Subtyping for session types in the pi calculus*. Gay and Hole.
[MSCS16] *Fair subtyping for multiparty sessions*. Padovani.

Output choices and role progress

- Related to *session subtyping*

```
global protocol Foo(role A, role B, role C) {
    choice at A { 1() from A to B; }
             or { 2() from A to C; }
    do Foo(A, B, C);
}
```

[ACTA05] *Subtyping for session types in the pi calculus*. Gay and Hole.
[MSCS16] *Fair subtyping for multiparty sessions*. Padovani.

Output choices and role progress

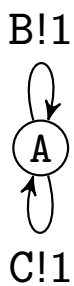
```
global protocol Foo(role A, role B, role C) {
    choice at A { 1() from A to B; }
             or { 2() from A to C; }
    do Foo(A, B, C);
}
```

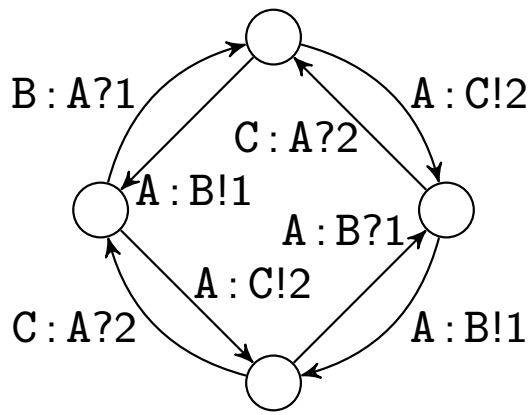
[ACTA05] *Subtyping for session types in the pi calculus*. Gay and Hole.
[MSCS16] *Fair subtyping for multiparty sessions*. Padovani.

- We implement two basic views:
  - Fair output choices (as modelled so far)
  - "Most unfair" – while still session type safe
    - Endpoints commit to a single case in any output choice (Extreme "naive" output choice subtyping)
    - Modelled by a transformation on endpoint FSMs

Output choices and role progress

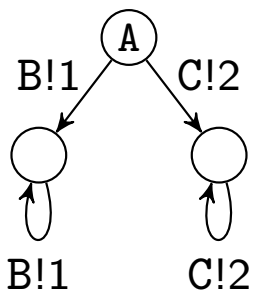
```
global protocol Foo(role A, role B, role C) {
    choice at A { 1() from A to B; }
             or { 2() from A to C; }
    do Foo(A, B, C);
}
```

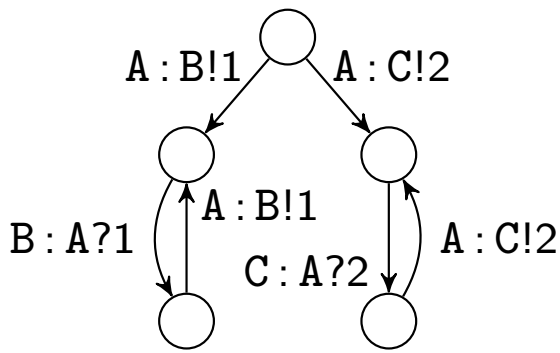




Output choices and role progress

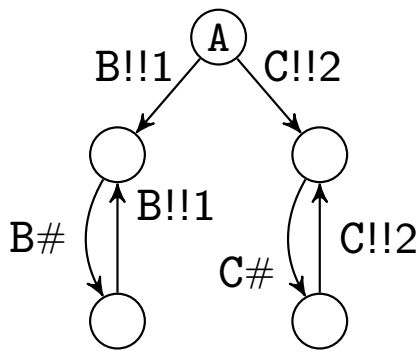
```
global protocol Foo(role A, role B, role C) {
    choice at A { 1() from A to B; }
             or { 2() from A to C; }
    do Foo(A, B, C);
}
```

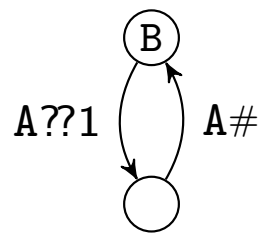




Output choices and role progress

```
explicit global protocol Foo(role A, role B, role C) {
    choice at A { 1() connect A to B; disconnect A and B; }
             or { 2() connect A to C; disconnect A and C; }
    do Foo(A, B, C);
}
```





Output choices and role progress

- OK if fairness assumed

```
global protocol Foo(role A, role B, role C) {
    choice at A {
        1() from A to B;
        do Foo(A, B, C);
    } or {
        2() from A to B;
        2() from B to C;
}   }
```

Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

  `connect A to B; .. connect A to C; .. connect B to C; ...`
- Travel Agency (`accpt` case) with dynamic port forwarding

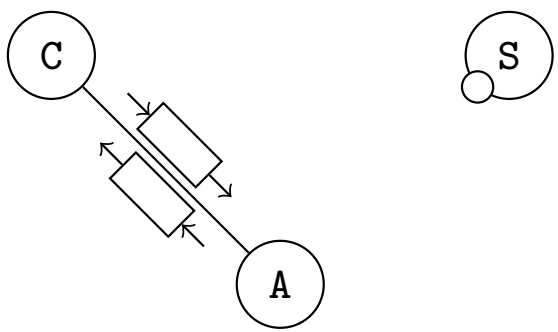
Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

```
connect A to B; .. connect A to C; .. connect B to C; ...
```

- Travel Agency (`accpt` case) with dynamic port forwarding

```
...
accpt() from C to A;
connect A to S;
port(Int) from S to A;
port(Int) from A to C;
connect C to S;
pay(Str) from C to S;
confirm(Int) from S to C;
...
```



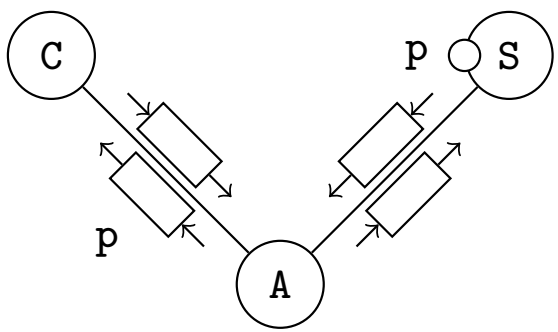
Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

```
connect A to B; .. connect A to C; .. connect B to C; ...
```

- Travel Agency (`accpt` case) with dynamic port forwarding

```
...
accpt() from C to A;
connect A to S;
port(Int) from S to A;
port(Int) from A to C;
connect C to S;
pay(Str) from C to S;
confirm(Int) from S to C;
...
```



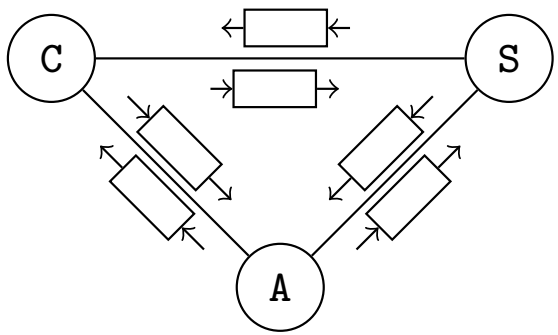
Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

```
connect A to B; .. connect A to C; .. connect B to C; ...
```

- Travel Agency (`accpt` case) with dynamic port forwarding

```
...
accpt() from C to A;
connect A to S;
port(Int) from S to A;
port(Int) from A to C;
connect C to S;
pay(Str) from C to S;
confirm(Int) from S to C;
...
```



Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

```
connect A to B; .. connect A to C; .. connect B to C; ...
```

- Travel Agency (`accpt` case) with dynamic port forwarding

```
...
accpt() from C to A;
connect A to S;
port(p:Int) from S to A; @"open=p:C"
port(p) from A to C;
connect C to S; @"port=p"
pay(Str) from C to S;
confirm(Int) from S to C;
...
```

[RV13] *Practical interruptible conversations*. Hu, Neykova, Yoshida, Demangeon and Honda.

Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

```
connect A to B; .. connect A to C; .. connect B to C; ...
```

- Travel Agency (`accpt` case) with dynamic port forwarding

```
...
accpt() from C to A;
connect A to S;
port(p:Int) from S to A; @"open=p:C"
//port(p) from A to C;
connect C to S; @"port=p"   ✗
pay(Str) from C to S;
confirm(Int) from S to C;
...
```

[RV13] *Practical interruptible conversations*. Hu, Neykova, Yoshida, Demangeon and Honda.

Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

```
connect A to B; .. connect A to C; .. connect B to C; ...
```

- Travel Agency (`accpt` case) with dynamic port forwarding

```
...
accpt() from C to A;
connect A to S;
port(p:Int) from S to A; @"open=p:C"
port(p) from A to C;
connect C to S; @"port=p"
pay(Str) from C to S;
confirm(Int) from S to C;
...
```

```
s_C.receive(A, port).connect(S, ..., host_S, pay, "payment details")...
s_A.receive(S, port).send(C, port)...
s_S.send(A, port).accept(C, pay, b)...
```

Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

```
connect A to B; .. connect A to C; .. connect B to C; ...
```

- Travel Agency (`accpt` case) with dynamic port forwarding

```
...
accpt() from C to A;
connect A to S;
port(p:Int) from S to A; @"open=p:C"
port(p) from A to C;
connect C to S; @"port=p"
pay(Str) from C to S;
confirm(Int) from S to C;
...
```

```
s_C.receive(A, port).connect(S, ..., host_S, pay, "payment details")...
s_A.receive(S, port).send(C, port)...
s_S.send(A, port).accept(C, pay, b)...
```

Multiparty correlation of binary connections

- Modelling based on a single session with one endpoint process per role
  - Connection mechanism (in particular, addressing) left abstract
- In practice: correlation by session identifier tags, port coordination, ...

```
connect A to B; .. connect A to C; .. connect B to C; ...
```

- Travel Agency (`accpt` case) with dynamic port forwarding

```
...
accpt() from C to A;
connect A to S;
port(p:Int) from S to A; @"open=p:C"
port(p) from A to C;
connect C to S; @"port=p"
pay(Str) from C to S;
confirm(Int) from S to C;
...
```

```
s_C.receive(A, port).connect(S, ..., host_S, pay, "payment details")...
s_A.receive(S, port).send(C, port)...
s_S.send(A, port).accept(C, pay, b)...
```

FTP (active/passive modes)

```
220 from S to C;
USER from C to S;
choice at S {
  331 from S to C;
  PASS from C to S;
  choice at S {
    230 from S to C;
    choice at C {
      PASV from C to S; // Passive mode
      choice at S {
        227(p:Int) from S to C; @"open=p:C"
        ...
      } or {
        ...
      }
    } or {
      PORT(q:Int) from C to S; @"open=q:S" // Active mode
      choice at S {
        200 from S to C;
        ...
      }
    } or {
      ...
} } }
```

Microservices use case

```
explicit global protocol InfoAuth
      (role LoginSvc, role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc,
       role ContractSvc) {
  connect Client to LoginSvc;
  login(UserName, password) from Client to LoginSvc;
  choice at LoginSvc {
    loginfailure() from LoginSvc to Client;
  } or {
    loginsuccess() from LoginSvc to Client;
    disconnect Client and LoginSvc;
    connect Client to AuthSvc;
    do Main(Client, AuthSvc, Filtersvc,
        SupplierSvc, ContractSvc);
} }

aux global protocol Main
      (role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc,
       role ContractSvc) {
  choice at Client {
    getsuppliers(UUID) from Client to AuthSvc;
    do SuppInfo(Client, AuthSvc,
        Filtersvc, SupplierSvc);
  } or {
    getcontracts() from Client to AuthSvc;
    do ContractInfo(Client, AuthSvc,
        Filtersvc, ContractSvc);
  }
  do Main(Client, AuthSvc, Filtersvc,
      SupplierSvc, ContractSvc);
}
```

```
aux global protocol SuppInfo
      (role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc) {
  choice at AuthSvc {
    deny() from AuthSvc to Client;
  } or {
    connect AuthSvc to SupplierSvc;
    getsuppliers() from AuthSvc to SupplierSvc;
    suppliers() from SupplierSvc to AuthSvc;
    disconnect AuthSvc and SupplierSvc;
    do FilterInfo
        <Filtersuppliers(UserContext,
            Filters, SupplierDetails)>
        (AuthSvc, Filtersvc);
    suppliers() from AuthSvc to Client;
  }
}

aux global protocol ContractInfo
      (role Client, role AuthSvc,
       role Filtersvc, role ContractSvc) {
  choice at AuthSvc {
    ...
} }

aux global protocol FilterInfo
      <sig Query>
      (role AuthSvc, role Filtersvc) {
  Query connect AuthSvc to Filtersvc;
  filtered() from Filtersvc to AuthSvc;
  disconnect AuthSvc and Filtersvc;
}
```

Microservices use case

```
explicit global protocol InfoAuth
      (role LoginSvc, role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc,
       role ContractSvc) {
  connect Client to LoginSvc;
  login(UserName, password) from Client to LoginSvc;
  choice at LoginSvc {
    loginfailure() from LoginSvc to Client;
  } or {
    loginsuccess() from LoginSvc to Client;
    disconnect Client and LoginSvc;
    connect Client to AuthSvc;
    do Main(Client, AuthSvc, Filtersvc,
        SupplierSvc, ContractSvc);
} }

aux global protocol Main
      (role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc,
       role ContractSvc) {
  choice at Client {
    getsuppliers(UUID) from Client to AuthSvc;
    do SuppInfo(Client, AuthSvc,
        Filtersvc, SupplierSvc);
  } or {
    getcontracts() from Client to AuthSvc;
    do ContractInfo(Client, AuthSvc,
        Filtersvc, ContractSvc);
  }
  do Main(Client, AuthSvc, Filtersvc,
      SupplierSvc, ContractSvc);
}
```

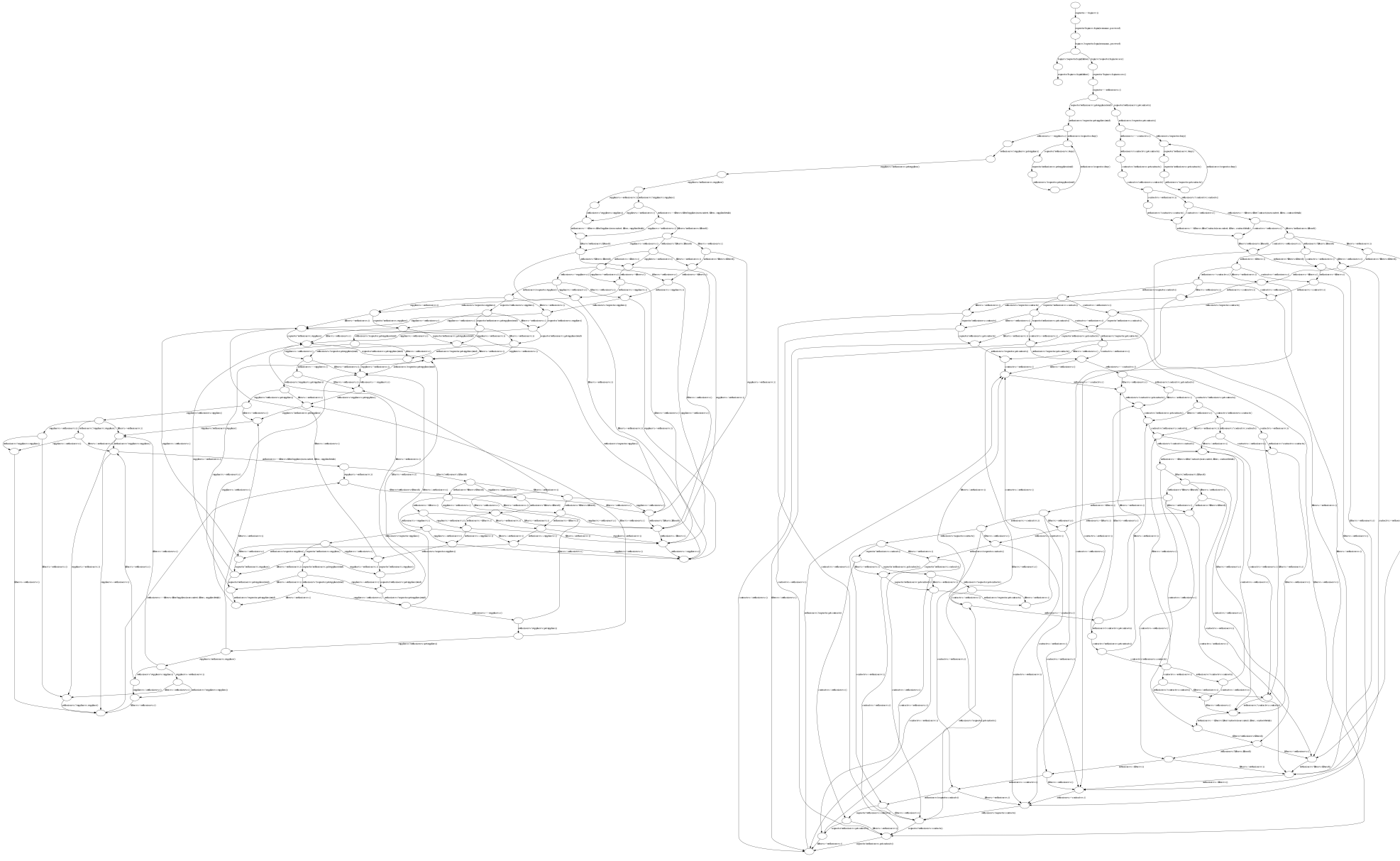
```
aux global protocol SuppInfo
      (role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc) {
  choice at AuthSvc {
    deny() from AuthSvc to Client;
  } or {
    connect AuthSvc to SupplierSvc;
    getsuppliers() from AuthSvc to SupplierSvc;
    suppliers() from SupplierSvc to AuthSvc;
    disconnect AuthSvc and SupplierSvc;
    do FilterInfo
        <Filtersuppliers(UserContext,
            Filters, SupplierDetails)>
        (AuthSvc, Filtersvc);
    suppliers() from AuthSvc to Client;
  }
}

aux global protocol ContractInfo
      (role Client, role AuthSvc,
       role Filtersvc, role ContractSvc) {
  choice at AuthSvc {
    ...
} }

aux global protocol FilterInfo
      <sig Query>
      (role AuthSvc, role Filtersvc) {
  Query connect AuthSvc to Filtersvc;
  filtered() from Filtersvc to AuthSvc;
  disconnect AuthSvc and Filtersvc;
}
```

Microservices use case

```
explicit global protocol InfoAuth
      (role LoginSvc, role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc,
       role ContractSvc) {
  connect Client to LoginSvc;
  login(UserName, password) from Client to LoginSvc;
  choice at LoginSvc {
    loginfailure() from LoginSvc to Client;
  } or {
    loginsuccess() from LoginSvc to Client;
    disconnect Client and LoginSvc;
    connect Client to AuthSvc;
    do Main(Client, AuthSvc, Filtersvc,
        SupplierSvc, ContractSvc);
} }

aux global protocol Main
      (role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc,
       role ContractSvc) {
  choice at Client {
    getsuppliers(UUID) from Client to AuthSvc;
    do SuppInfo(Client, AuthSvc,
        Filtersvc, SupplierSvc);
  } or {
    getcontracts() from Client to AuthSvc;
    do ContractInfo(Client, AuthSvc,
        Filtersvc, ContractSvc);
  }
  do Main(Client, AuthSvc, Filtersvc,
      SupplierSvc, ContractSvc);
}
```

```
aux global protocol SuppInfo
      (role Client, role AuthSvc,
       role Filtersvc, role SupplierSvc) {
  choice at AuthSvc {
    deny() from AuthSvc to Client;
  } or {
    connect AuthSvc to SupplierSvc;
    getsuppliers() from AuthSvc to SupplierSvc;
    suppliers() from SupplierSvc to AuthSvc;
    disconnect AuthSvc and SupplierSvc;
    do FilterInfo
        <Filtersuppliers(UserContext,
            Filters, SupplierDetails)>
        (AuthSvc, Filtersvc);
    suppliers() from AuthSvc to Client;
  }
}

aux global protocol ContractInfo
      (role Client, role AuthSvc,
       role Filtersvc, role ContractSvc) {
  choice at AuthSvc {
    ...
} }

aux global protocol FilterInfo
      <sig Query>
      (role AuthSvc, role Filtersvc) {
  Query connect AuthSvc to Filtersvc;
  filtered() from Filtersvc to AuthSvc;
  disconnect AuthSvc and Filtersvc;
}
```

Microservices use case



Related work

- Dynamic participation in sessions/conversations

  [ESOP09] *Conversation types*. Caires and Vieira.
  [POPL11] *Dynamic multirole session types*. Deniélou and Yoshida.
  [CONCUR12] *Nested protocols in session types*. Demangeon and Honda.

- Dynamic message sequence charts and communication automata

  [FSTTCS02] *Dynamic message sequence charts*. Leucker, Madhusudan and Mukhopadhyay.
  [LATA13] *Dynamic communication automata and branching high-level MSCs*. Bollig, Cyriac, Hélou et, Jara and Schwentick.

- CFSM-based well-formedness of choreographies and MPST

  [POPL08] *Deciding choreography realizability*. Basu and Bultan.
  [ICALP13] *Multiparty Compatibility in Communicating Automata*. Deniélou and Yoshida.
  [CONCUR15] *Meeting Deadlines Together*. Bocchi, Lange and Yoshida.
  [PLACES16] *Multiparty compatibility for concurrent objects*. Perera, Lange and Gay.

- Implementations of session types

  Java ([ECOOP08,SCP13,PPDP16,FASE16]), Scala ([ECOOP16,ECOOP17]), Haskell ([PADL04,HASKELL08,PLACES10,POPL16,HASKELL16]), OCaml ([JFP17,ESOP17,COORDINATION17]), SILL ([ESOP13,FoSSaCS15]), Links ([ESOP15]), Python ([RV13]), Rust ([WGP15]), C ([TOOLS12]), ...

Conclusions and future work

- (We can finally do Travel Agency in MPST!)
- Practically-motivated extension for explicit connection actions in MPST
  - Scribble toolchain for MPST validation and Endpoint API generation
    - Integrating MPST with existing model checking techniques and tools

      [TACAS16] *Characteristic Formulae for Session Types*. Lange and Yoshida.
    - (The session type system – interplay with delegation)
  - Other kinds of communication actions? *e.g.*, SSL/TLS connection wrapping (HTTPS, SMTP, FTPS, ...)
  - Integration of further extensions from MPST theory
    - *e.g.*, time, asynchronous interrupts, nested subsessions, message value assertions, role parameterisations, event handling, ...
- Thanks!
  - `https://github.com/scribble/scribble-java`
  - `https://www.doc.ic.ac.uk/~rhu/scribble/explicit.html`