Statically Verified Refinements for Multiparty Protocols

Fangyi Zhou, Francisco Ferreira, Raymond Hu, Rumyana Neykova, Nobuko Yoshida

Imperial College London
University of Hertfordshire UH
Brunel University London

SPLASH 2020 - OOPSLA - November 2020

Statically Verified Refinements for Multiparty Protocols

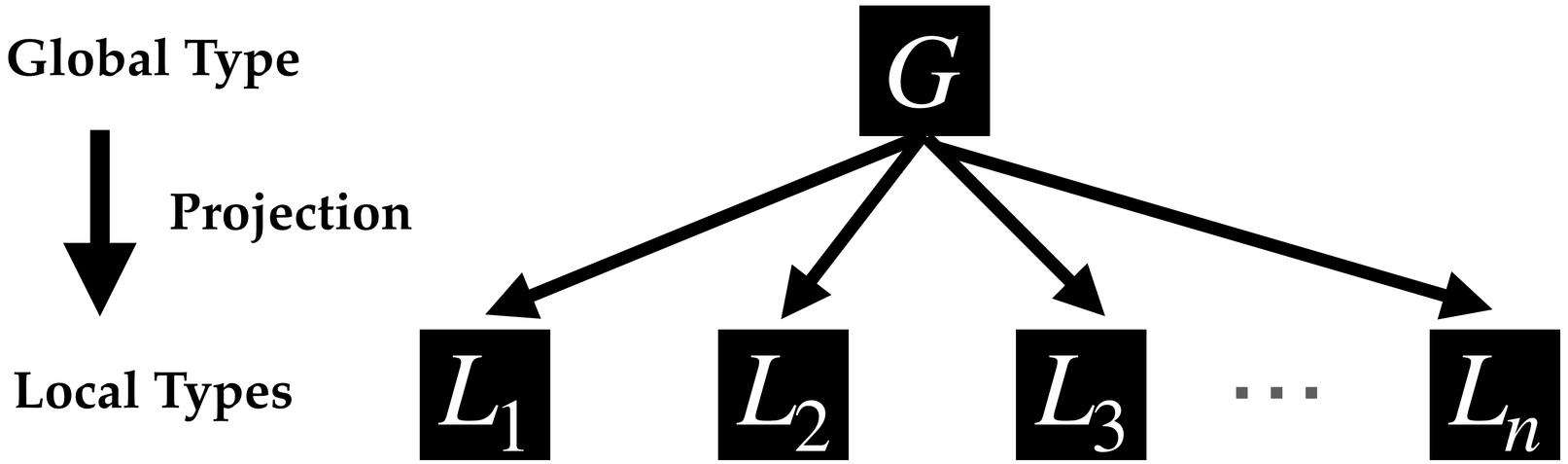
Multiparty Session Types (MPST)

A typing discipline for message passing concurrency

Global Type

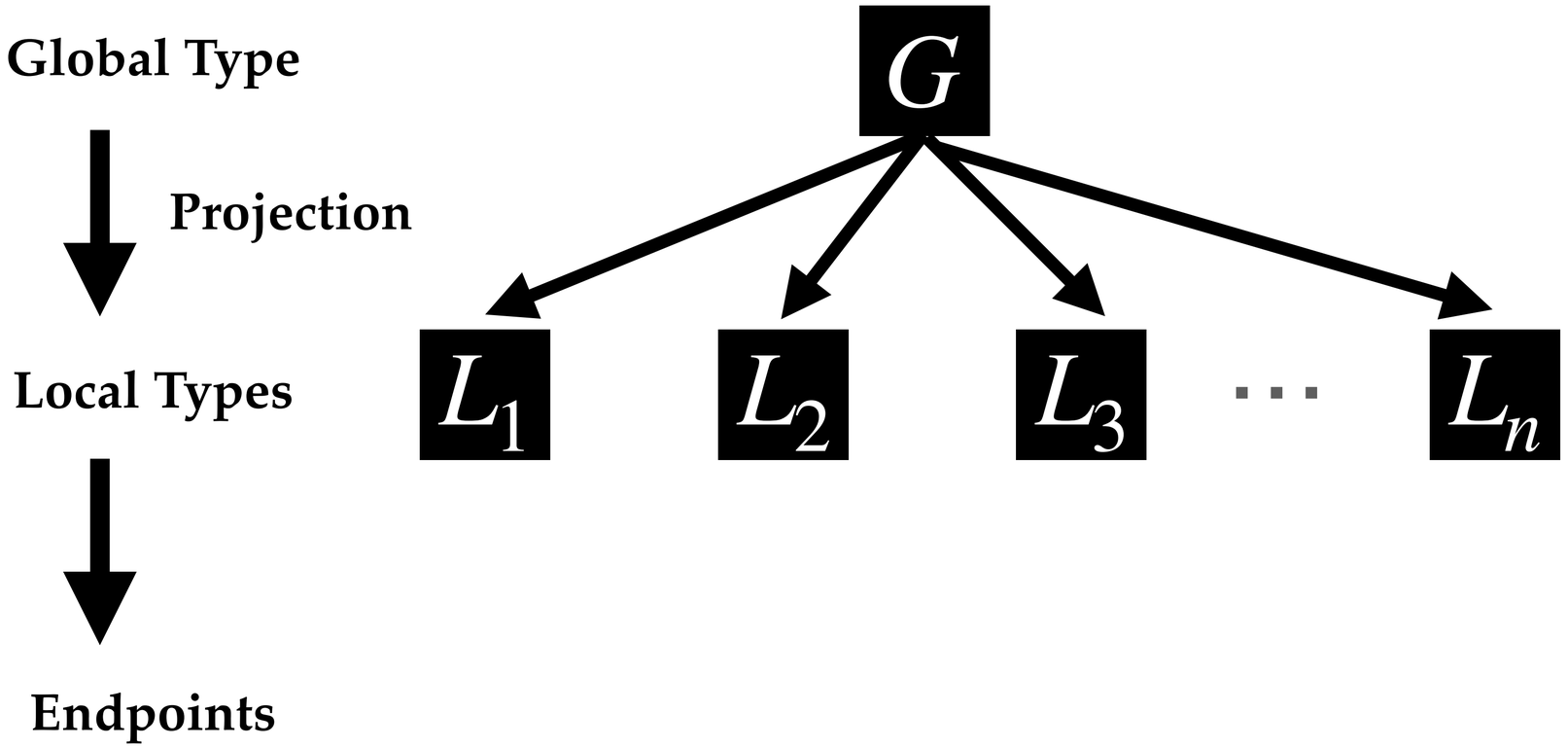
Multiparty Session Types (MPST)

A typing discipline for message passing concurrency



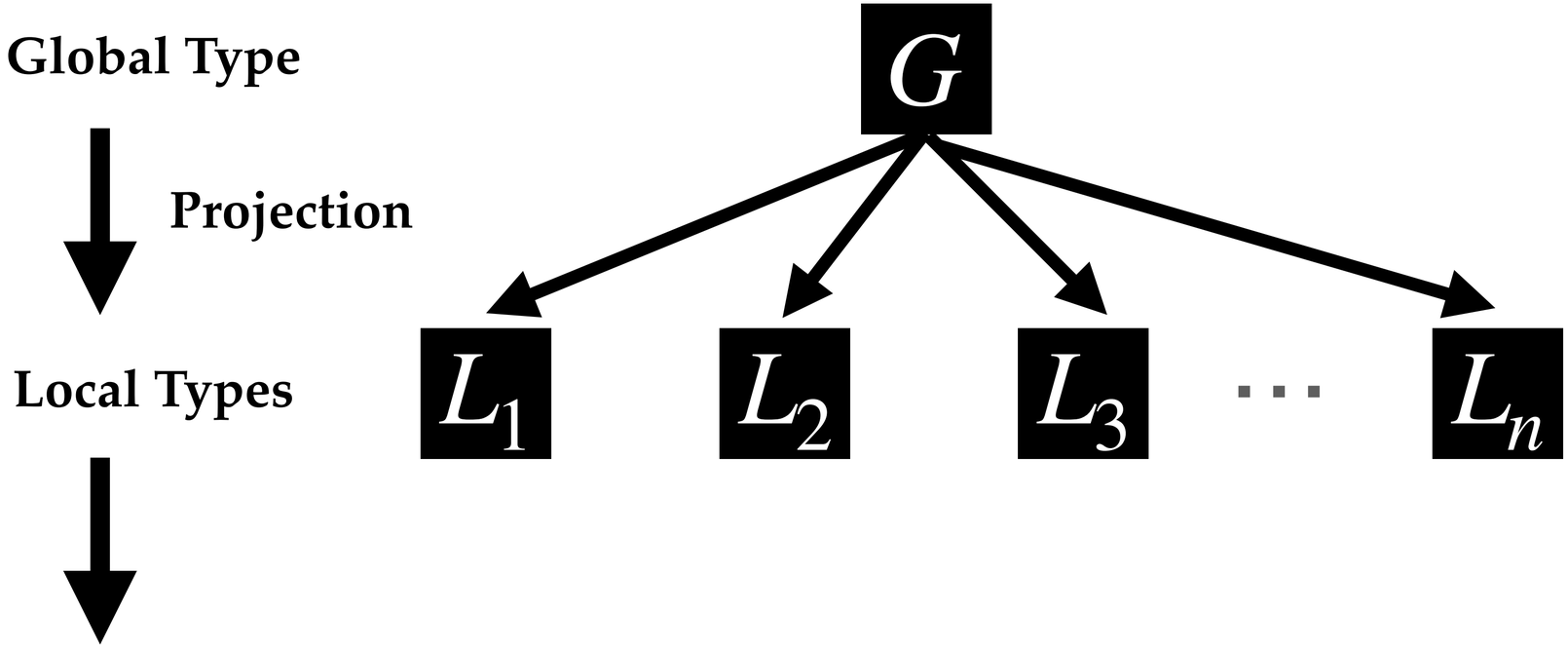
Multiparty Session Types (MPST)

A typing discipline for message passing concurrency



Multiparty Session Types (MPST)

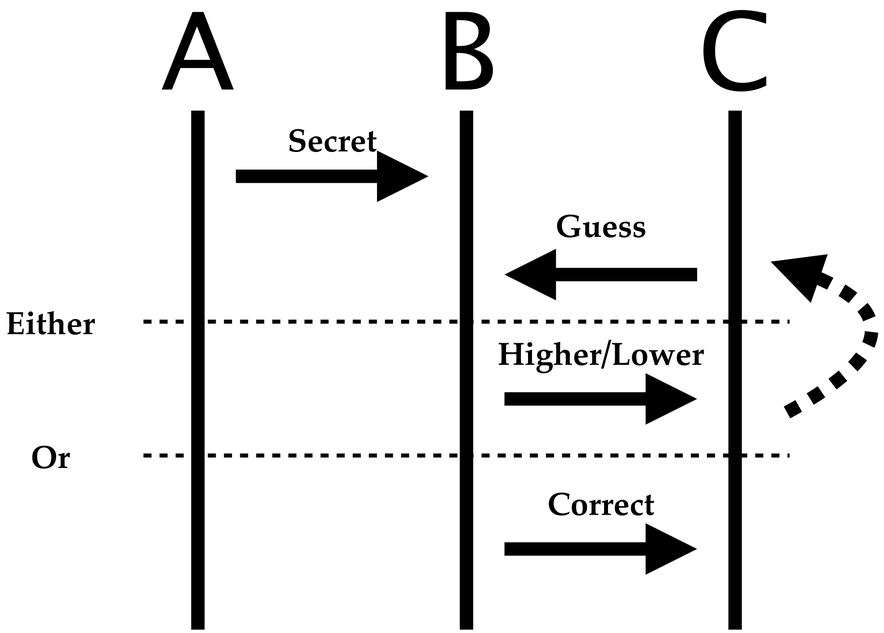
A typing discipline for message passing concurrency



Free from deadlocks or communication mismatches

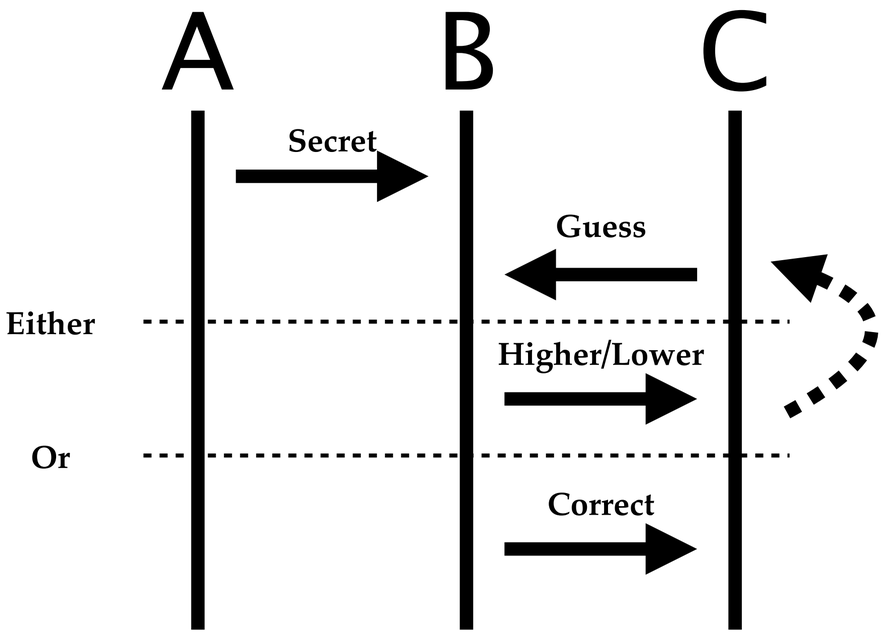
Multiparty Session Types (MPST)

A typing discipline for message passing concurrency



Multiparty Session Types (MPST)

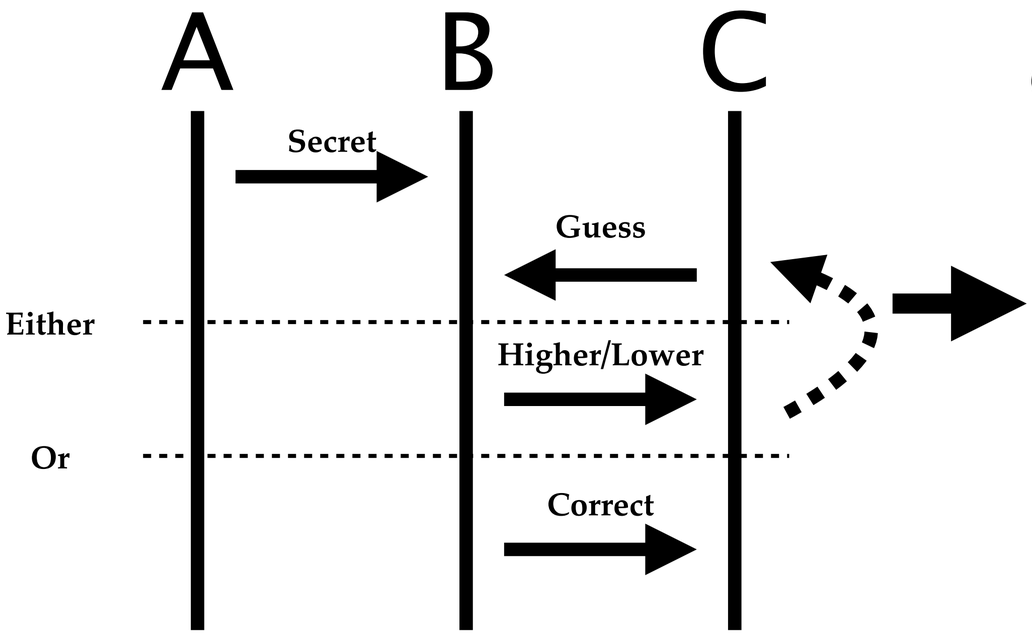
A typing discipline for message passing concurrency



$$G = \mathsf{A} \to \mathsf{B} : \mathit{Secret}(\texttt{int}) .$$
$$\mu \mathbf{t} . \mathsf{C} \to \mathsf{B} : \mathit{Guess}(\texttt{int}) .$$
$$\mathsf{B} \to \mathsf{C}$$
$$\left\{ \begin{array}{l} \mathit{Higher}() . \mathbf{t} \\ \mathit{Lower}() . \mathbf{t} \\ \mathit{Correct}() . \texttt{end} \end{array} \right\}$$

Multiparty Session Types (MPST)

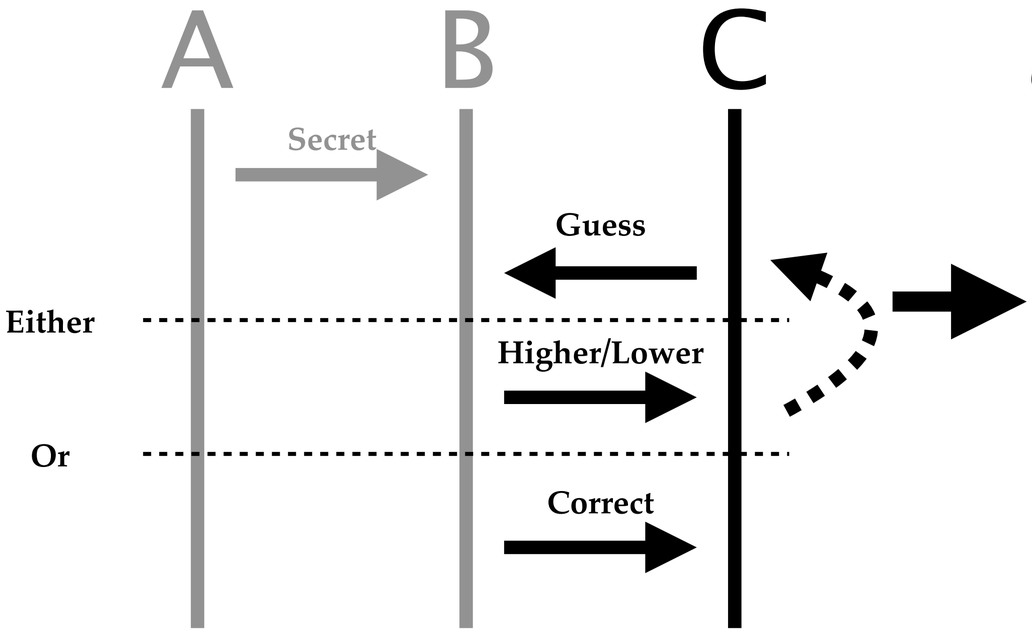
A typing discipline for message passing concurrency



$$G = \mathsf{A} \to \mathsf{B} : \mathit{Secret}(\texttt{int}) .$$
$$\mu \mathbf{t} . \mathsf{C} \to \mathsf{B} : \mathit{Guess}(\texttt{int}) .$$
$$\mathsf{B} \to \mathsf{C}$$
$$\left\{ \begin{array}{l} \mathit{Higher}() . \mathbf{t} \\ \mathit{Lower}() . \mathbf{t} \\ \mathit{Correct}() . \texttt{end} \end{array} \right\}$$

Multiparty Session Types (MPST)

A typing discipline for message passing concurrency



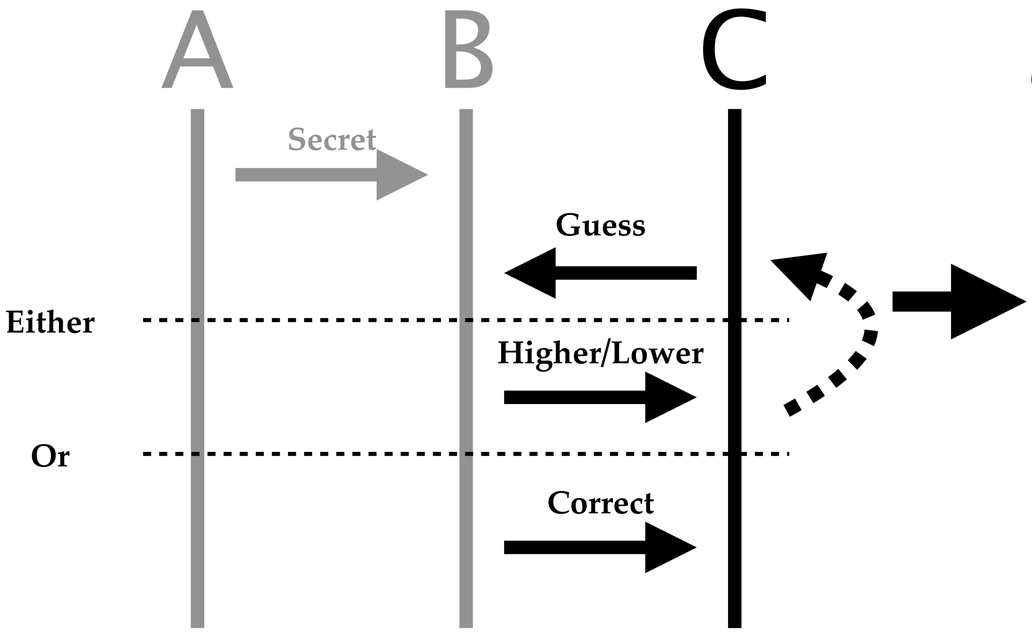
$$G = \mathsf{A} \to \mathsf{B} : \mathit{Secret}(\mathtt{int}) .\ \mu \mathbf{t} . \mathsf{C} \to \mathsf{B} : \mathit{Guess}(\mathtt{int}) .\ \mathsf{B} \to \mathsf{C} \left\{ \begin{array}{l} \mathit{Higher}() . \mathbf{t} \\ \mathit{Lower}() . \mathbf{t} \\ \mathit{Correct}() . \mathtt{end} \end{array} \right\}$$

Projection

$$G \upharpoonright \mathsf{C} = \mu \mathbf{t} . \mathsf{B} \oplus \mathit{Guess}(\mathtt{int}) .\ \mathsf{B} \,\& \left\{ \begin{array}{l} \mathit{Higher}() . \mathbf{t} \\ \mathit{Lower}() . \mathbf{t} \\ \mathit{Correct}() . \mathtt{end} \end{array} \right\}$$

Multiparty Session Types (MPST)

A typing discipline for message passing concurrency



$$G = \mathsf{A} \to \mathsf{B} : \mathit{Secret}(\mathtt{int}) .\ \mu \mathbf{t} . \mathsf{C} \to \mathsf{B} : \mathit{Guess}(\mathtt{int}) .\ \mathsf{B} \to \mathsf{C} \left\{ \begin{array}{l} \mathit{Higher}() . \mathbf{t} \\ \mathit{Lower}() . \mathbf{t} \\ \mathit{Correct}() . \mathtt{end} \end{array} \right\}$$

$$G \upharpoonright \mathsf{C} = \mu \mathbf{t} . \mathsf{B} \oplus \mathit{Guess}(\mathtt{int}) .\ \mathsf{B} \,\& \left\{ \begin{array}{l} \mathit{Higher}() . \mathbf{t} \\ \mathit{Lower}() . \mathbf{t} \\ \mathit{Correct}() . \mathtt{end} \end{array} \right\}$$

Internal Choice

External Choice

Statically Verified Refinements for Multiparty Protocols

Refinement Types

Refining base types with predicates

All integers $x : \{\texttt{int} \mid \texttt{true}\}$

Natural numbers $x : \{\texttt{int} \mid x \geq 0\}$

Even natural numbers $x : \{\texttt{int} \mid x \geq 0 \ \wedge \ x \bmod 2 = 0\}$

Nothing $x : \{\texttt{int} \mid \texttt{false}\}$

Statically Verified Refinements for Multiparty Protocols

F* — Our Target Language

✓ Refinement / dependent types

✓ Code extraction to OCaml

✓ Automated verification via Z3

Contribution

- A theory combining MPST and refinement types

- A toolchain for implementing refined multiparty protocols

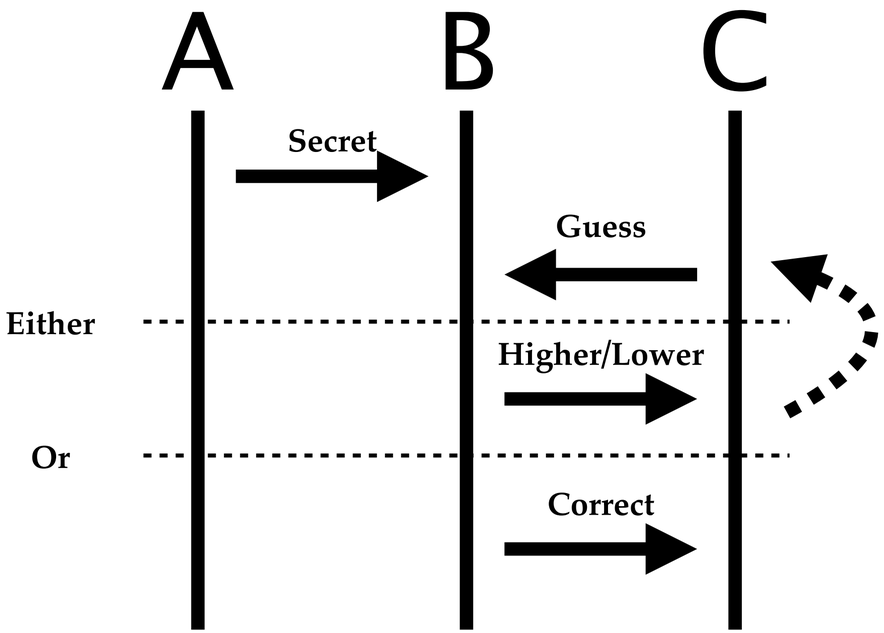
Contribution

- **A theory combining MPST and refinement types**

- A toolchain for implementing refined multiparty protocols

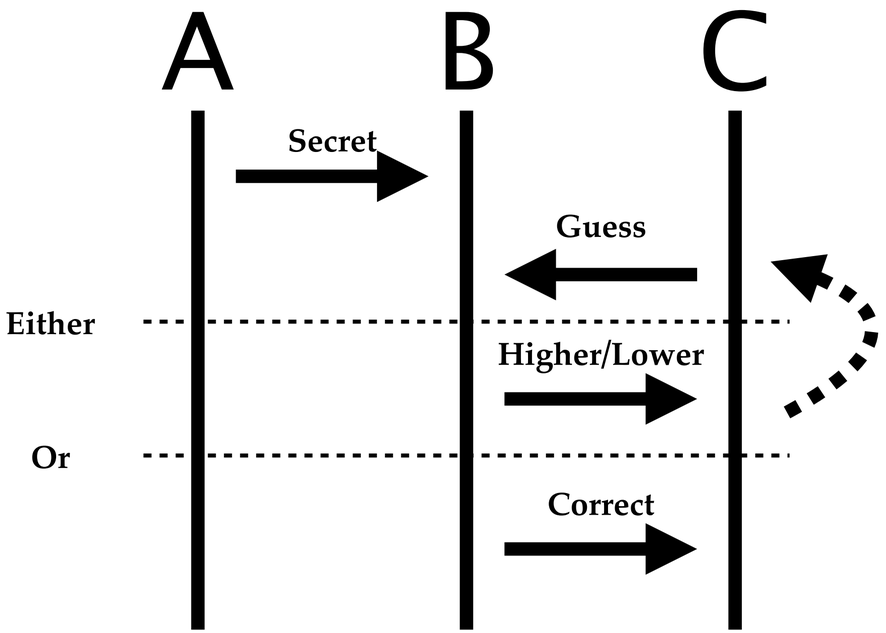
Refined Multiparty Session Types (RMPST)

A number guessing game



Refined Multiparty Session Types (RMPST)

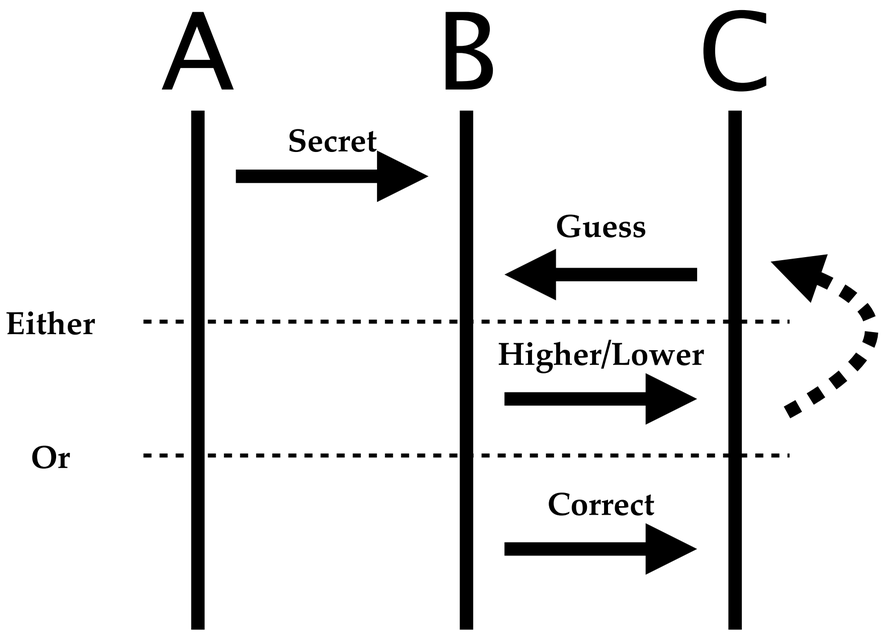
Combining refinement types and multiparty session types



$$G = \mathsf{A} \to \mathsf{B} : \mathit{Secret}(\texttt{int}) .$$
$$\mu \mathbf{t} . \mathsf{C} \to \mathsf{B} : \mathit{Guess}(\texttt{int}) .$$
$$\mathsf{B} \to \mathsf{C}$$
$$\left\{ \begin{array}{l} \mathit{Higher}() . \mathbf{t} \\ \mathit{Lower}() . \mathbf{t} \\ \mathit{Correct}() . \texttt{end} \end{array} \right\}$$

Refined Multiparty Session Types (RMPST)

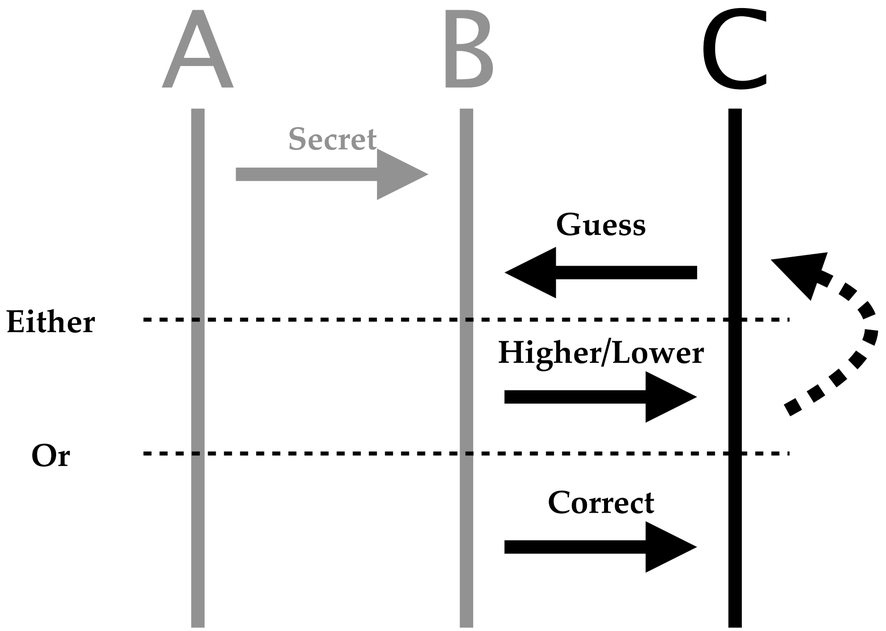
Combining refinement types and multiparty session types



$$G = \mathsf{A} \to \mathsf{B} : \mathit{Secret}(s : \mathtt{int}) .$$
$$\mu \mathbf{t} . \mathsf{C} \to \mathsf{B} : \mathit{Guess}(g : \mathtt{int}) .$$
$$\mathsf{B} \to \mathsf{C}$$
$$\left\{ \begin{array}{l} \mathit{Higher}(\{g < s\}) . \mathbf{t} \\ \mathit{Lower}(\{g > s\}) . \mathbf{t} \\ \mathit{Correct}(\{g = s\}) . \mathtt{end} \end{array} \right\}$$

Refined Multiparty Session Types (RMPST)

Challenge: Projection onto C



$$G = \mathsf{A} \to \mathsf{B} : \mathit{Secret}(s : \mathtt{int}) .$$
$$\mu \mathbf{t} . \mathsf{C} \to \mathsf{B} : \mathit{Guess}(g : \mathtt{int}) .$$
$$\mathsf{B} \to \mathsf{C}$$
$$\left\{ \begin{array}{l} \mathit{Higher}(\{g < s\}) . \mathbf{t} \\ \mathit{Lower}(\{g > s\}) . \mathbf{t} \\ \mathit{Correct}(\{g = s\}) . \mathtt{end} \end{array} \right\}$$

Tracking Knowledge in Typing Contexts

Globally, we track which participants know which variables

$$\Gamma ::= \cdot \mid \Gamma, x^{\mathbb{P}} : T$$

Tracking Knowledge in Typing Contexts

Globally, we track which participants know which variables

$$\Gamma ::= \cdot \mid \Gamma, x^{\mathbb{P}} : T$$

Denotes the set of participants knowing the variable

Tracking Knowledge in Typing Contexts

Globally, we track which participants know which variables

$$\Gamma ::= \cdot \mid \Gamma, x^{\mathbb{P}} : T$$

Denotes the set of participants knowing the variable

Locally, we track whether the value of a variable will be known

Tracking Knowledge in Typing Contexts

Globally, we track which participants know which variables

$$\Gamma ::= \cdot \mid \Gamma, x^{\mathbb{P}} : T$$

Denotes the set of participants knowing the variable

Locally, we track whether the value of a variable will be known

$$\Sigma ::= \cdot \mid \Sigma, x^{\theta} : T$$

Tracking Knowledge in Typing Contexts

Globally, we track which participants know which variables

$$\Gamma ::= \cdot \mid \Gamma, x^{\mathbb{P}} : T$$

Denotes the set of participants knowing the variable

Locally, we track whether the value of a variable will be known

$$\Sigma ::= \cdot \mid \Sigma, x^{\theta} : T$$

Denotes the multiplicity of the variable

Tracking Knowledge in Typing Contexts

Globally, we track which participants know which variables

$$\Gamma ::= \cdot \mid \Gamma, x^{\mathbb{P}} : T$$

Denotes the set of participants knowing the variable

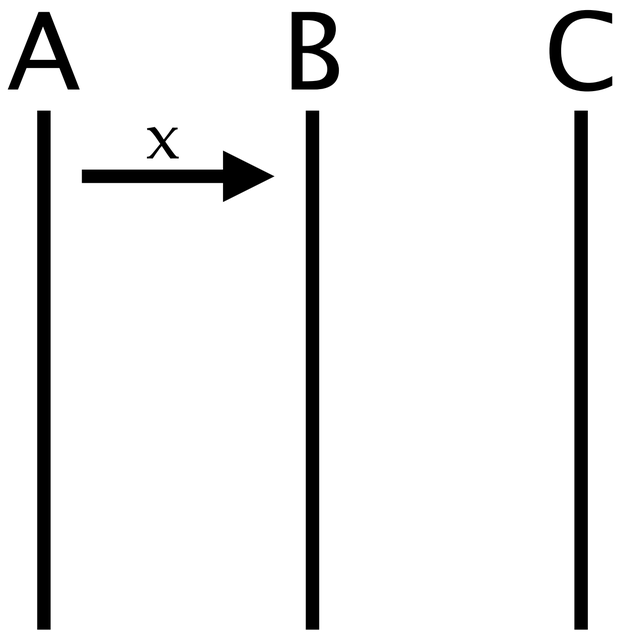
Locally, we track whether the value of a variable will be known

$$\Sigma ::= \cdot \mid \Sigma, x^{\theta} : T$$

Denotes the multiplicity of the variable

$$\theta ::= 0 \mid \omega$$

Irrelevant and Unrestricted Variables

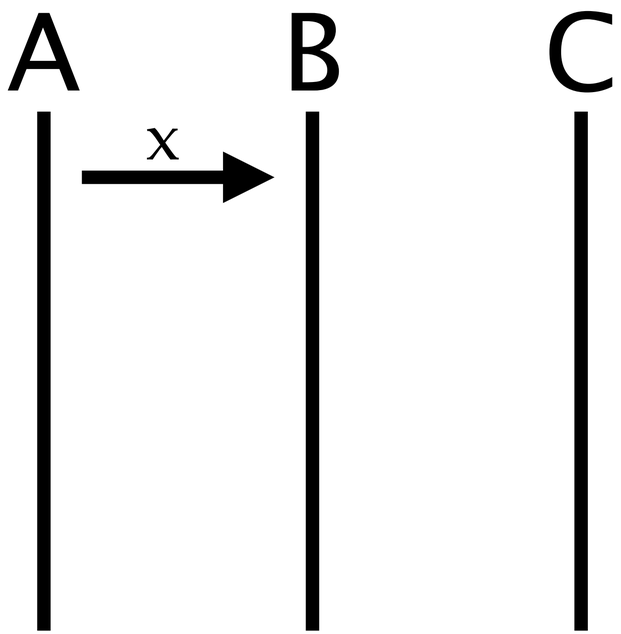


A: sends the variable x

B: receives the variable x

C: only knows about the existence of x

Irrelevant and Unrestricted Variables



A: sends the variable x^{ω}

B: receives the variable x^{ω}

C: only knows about the existence of x^{0}

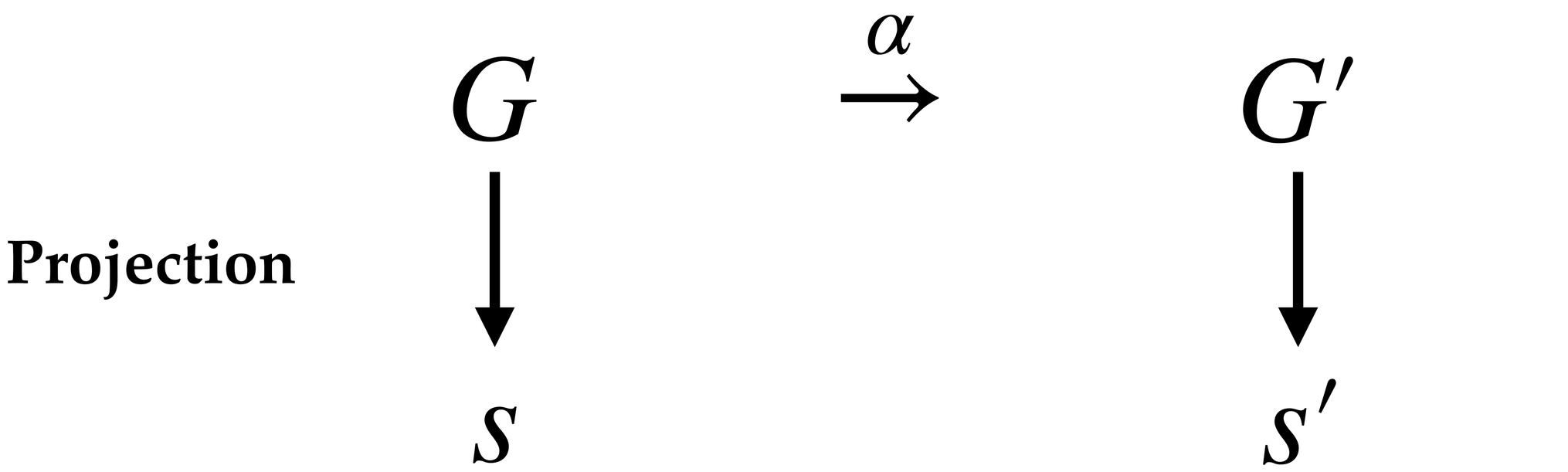
Semantics - Labelled Transition System

In the original theory

$$G \xrightarrow{\alpha} G'$$

Semantics - Labelled Transition System

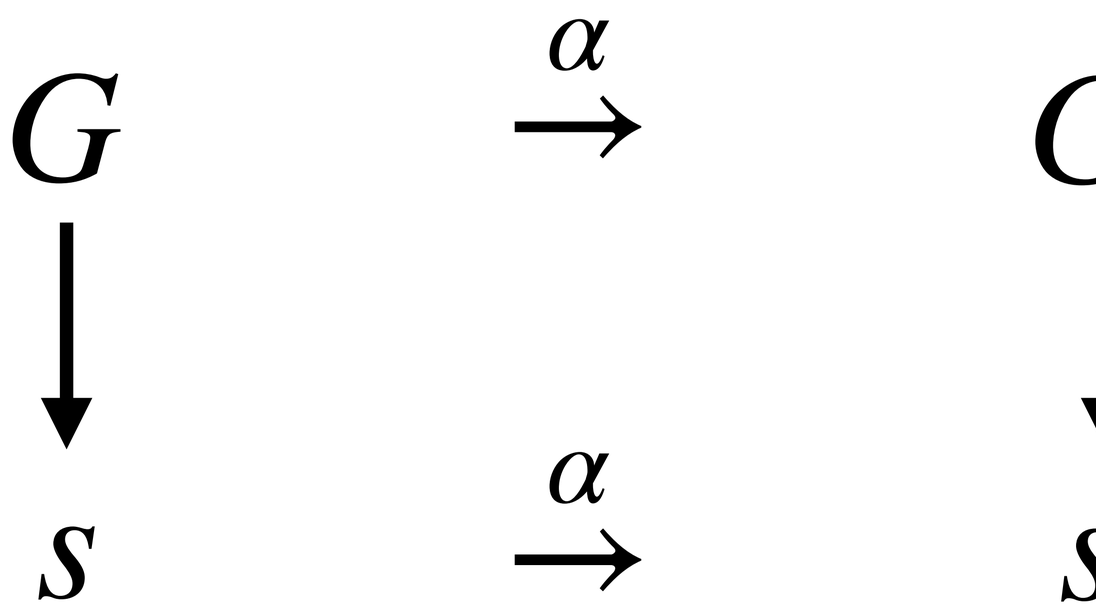
In the original theory



$$s = \{G \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G} \qquad s' = \{G' \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G'}$$

Semantics - Labelled Transition System

In the original theory



$$G \xrightarrow{\alpha} G'$$

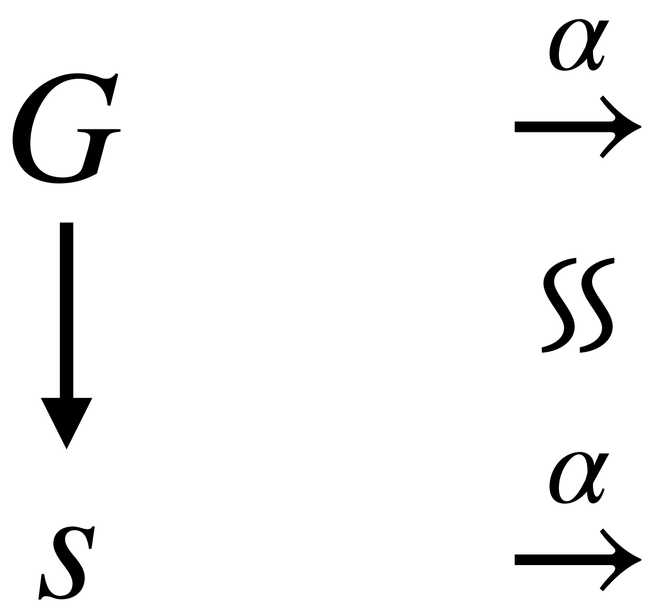
$$\downarrow \qquad \downarrow$$

$$s \xrightarrow{\alpha} s'$$

$$s = \{G \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G} \qquad s' = \{G' \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G'}$$

Semantics - Labelled Transition System

In the original theory



$$s = \{G \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G} \qquad s' = \{G' \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G'}$$

Semantics - Labelled Transition System

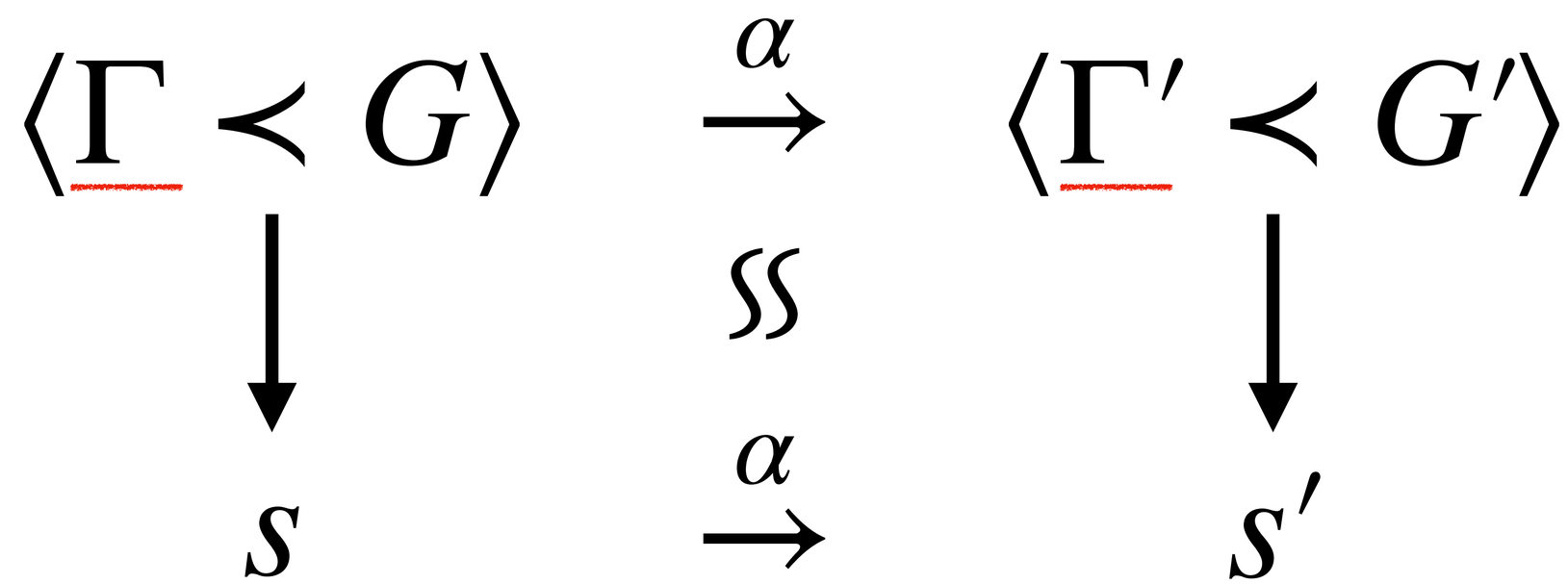
Our new theory

$$\begin{array}{ccc} \langle \Gamma \prec G \rangle & \xrightarrow{\alpha} & \langle \Gamma' \prec G' \rangle \\ \downarrow & \wr\wr & \downarrow \\ s & \xrightarrow{\alpha} & s' \end{array}$$

$$s = \{\langle \Gamma \prec G \rangle \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G} \qquad s' = \{\langle \Gamma' \prec G' \rangle \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G'}$$

Semantics - Labelled Transition System

Evolving contexts model the evolving knowledge obtained during reduction



$$s = \{\langle \Gamma \prec G \rangle \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G} \qquad s' = \{\langle \Gamma' \prec G' \rangle \upharpoonright \mathsf{r}\}_{\mathsf{r} \in G'}$$

Contribution

- A theory combining MPST and refinement types

- A toolchain for implementing refined multiparty protocols

Contribution

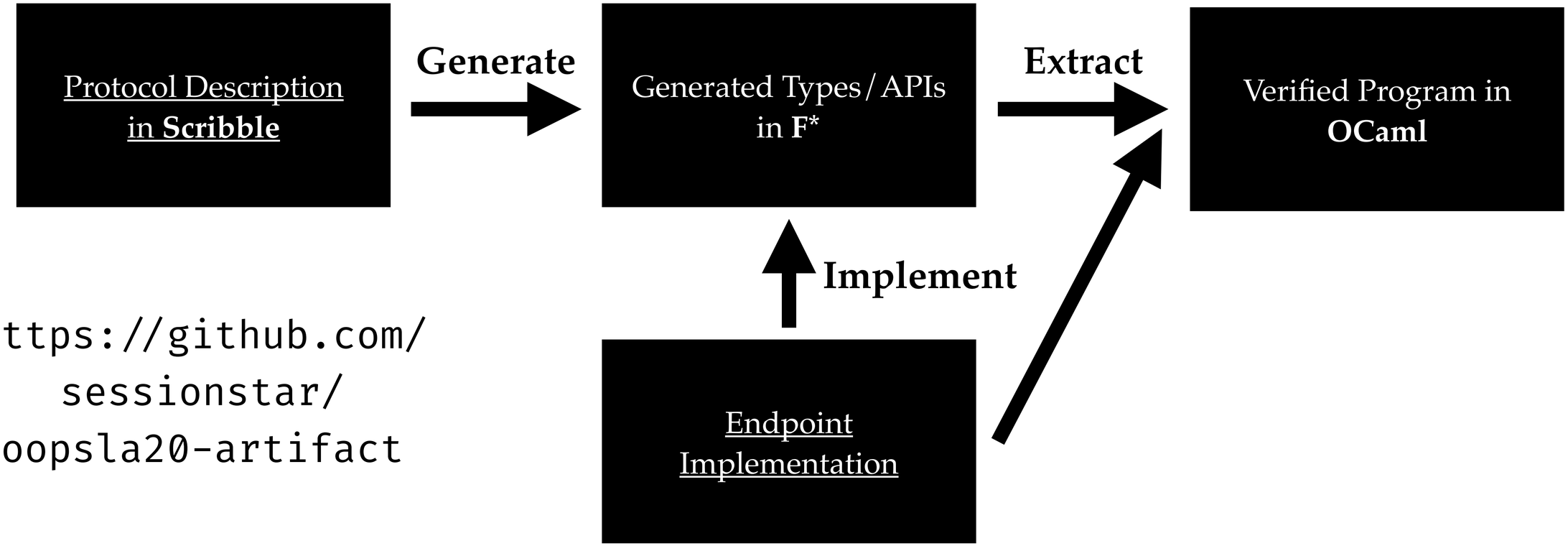
- A theory combining MPST and refinement types

- **A toolchain for implementing refined multiparty protocols**

Session* - A Toolchain for RMPST

Workflow

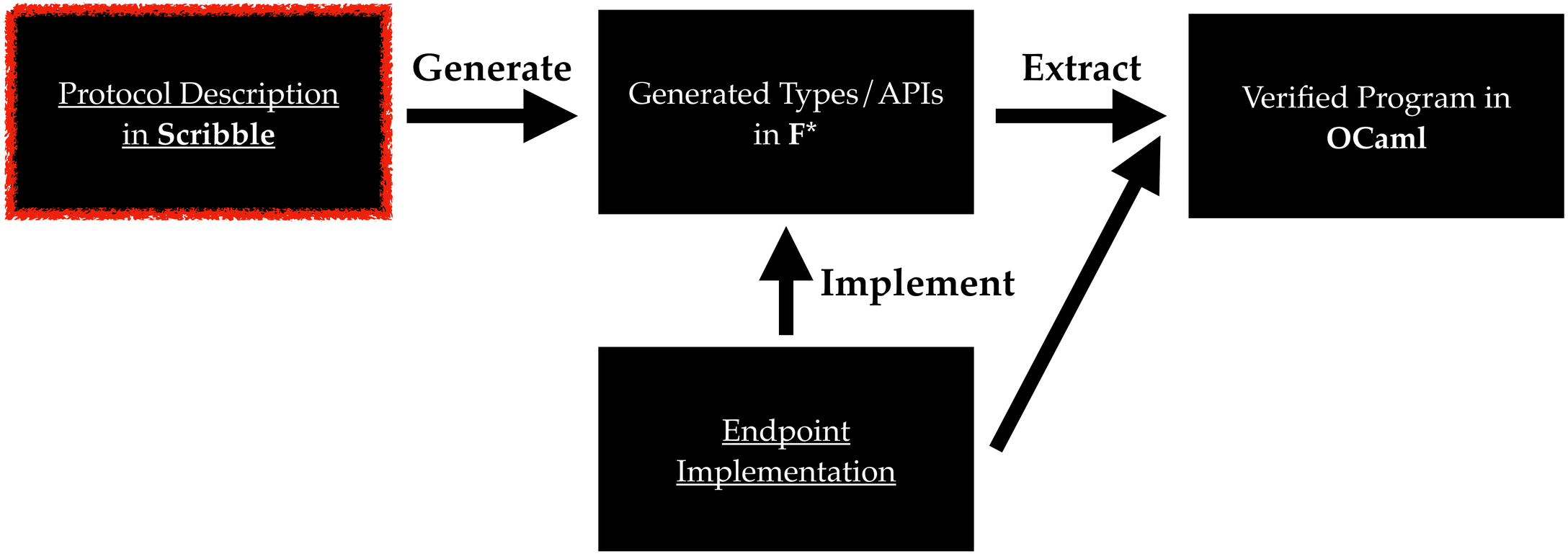
Scribble + F* = Safe Distributed Programming



https://github.com/sessionstar/oopsla20-artifact

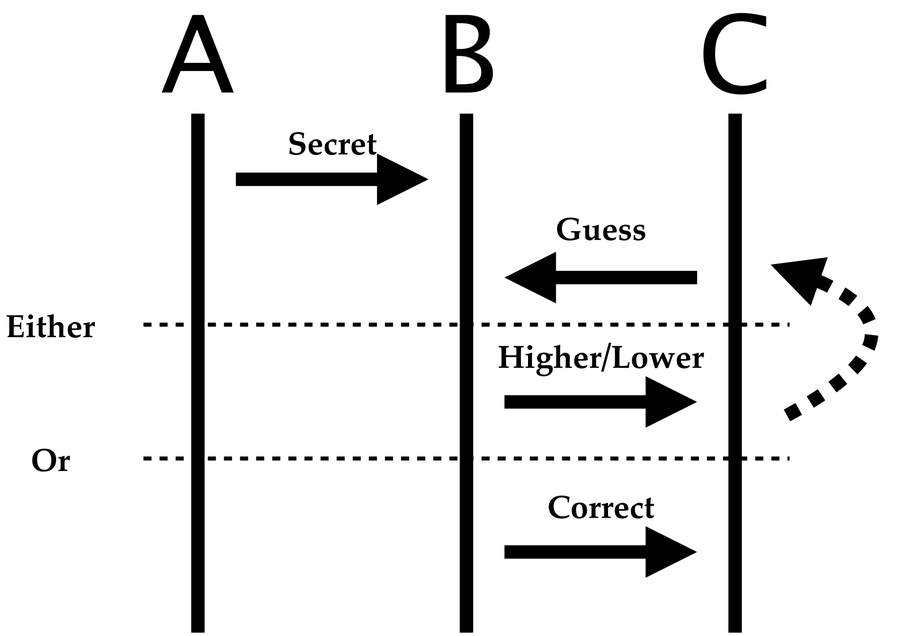
Session* - A Toolchain for RMPST

Workflow



Specifying Protocols

using the Scribble Protocol Description Language

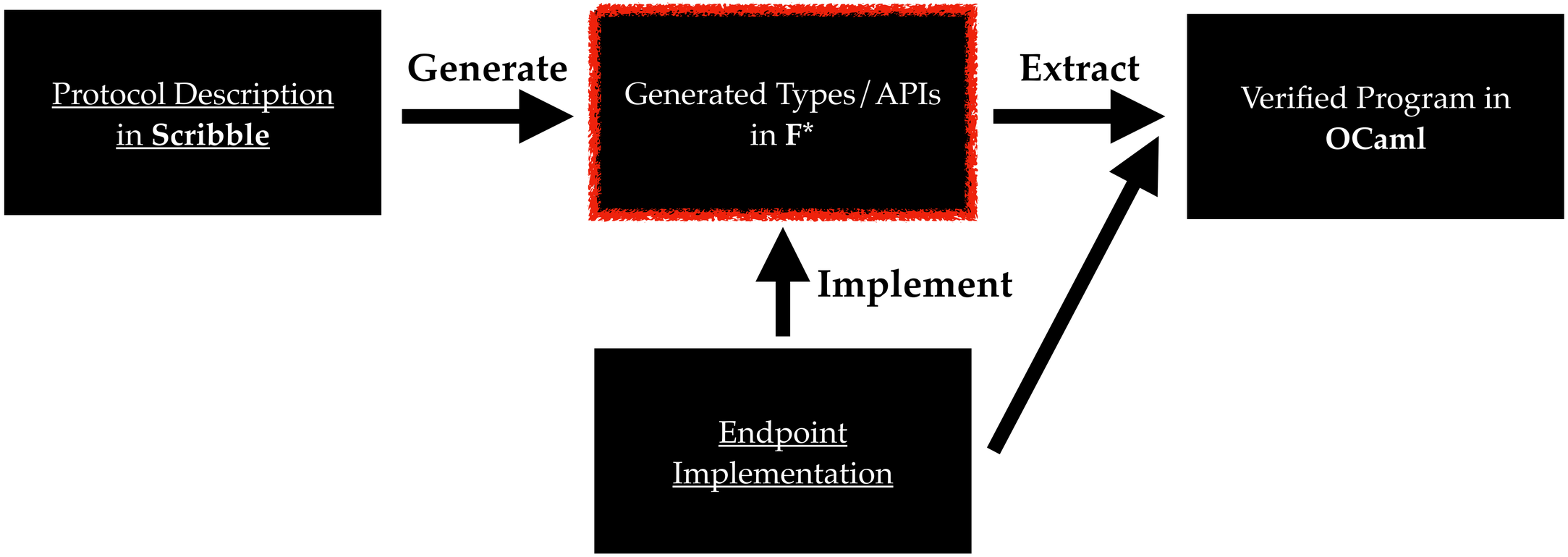



```
global protocol Guessing(role A, role B, role C)
{ Secret(s: int) from A to B;
  do Aux(A, B, C); @“B[s]” }

aux global protocol Aux(role A, role B, role C)
  @“B[s: int]”
{ Guess(g: int) from C to B;
  choice at B
     { Higher() from B to C; @“g < s”
       do Aux(A, B, C); @“B[s]” }
  or { Correct() from B to C; @“g = s” }
  or { Lower … } }
```

Session* - A Toolchain for RMPST

Workflow



API Style

Traditional

```
let s = connect () in
let (s, secr) = recv_Secret s in
let (s, guess) = recv_Guess s in
if guess = secr
then (let s = send_Correct s in
      close s)
else …
```

API Style

Traditional

```
let s = connect () in
let (s, secr) = recv_Secret s in
let (s, guess) = recv_Guess s in
if guess = secr
then (let s = send_Correct s in
      close s)
else …
```

Hard to ensure linearity!

API Style

Traditional

```
let s = connect () in
let (s, secr) = recv_Secret s in
let (s, guess) = recv_Guess s in
if guess = secr
then (let s = send_Correct s in
      close s)
else …
```

Hard to ensure linearity!

Callback Style (new)

For the internal choice of B

```
type choice (st: state) =
| Correct of unit{st.g = st.s}
| Higher of unit{st.g < st.s}
| Lower of unit{st.g > st.s}

type choice_send = (st: state) → choice st
```

API Style

Traditional

```
let s = connect () in
let (s, secr) = recv_Secret s in
let (s, guess) = recv_Guess s in
if guess = secr
then (let s = send_Correct s in
      close s)
else …
```

Hard to ensure linearity!

Callback Style (new)

For the external choice of C

```
type receive_Correct = (st: state)
     → unit{st.g = st.s} → unit
type receive_Higher = (st: state)
     → unit{st.g < st.s} → unit
type receive_Lower = (st: state)
     → unit{st.g > st.s} → unit
```

API Style

Traditional

```
let s = connect () in
let (s, secr) = recv_Secret s in
let (s, guess) = recv_Guess s in
if guess = secr
then (let s = send_Correct s in
      close s)
else …
```

Hard to ensure linearity!

Callback Style (new)

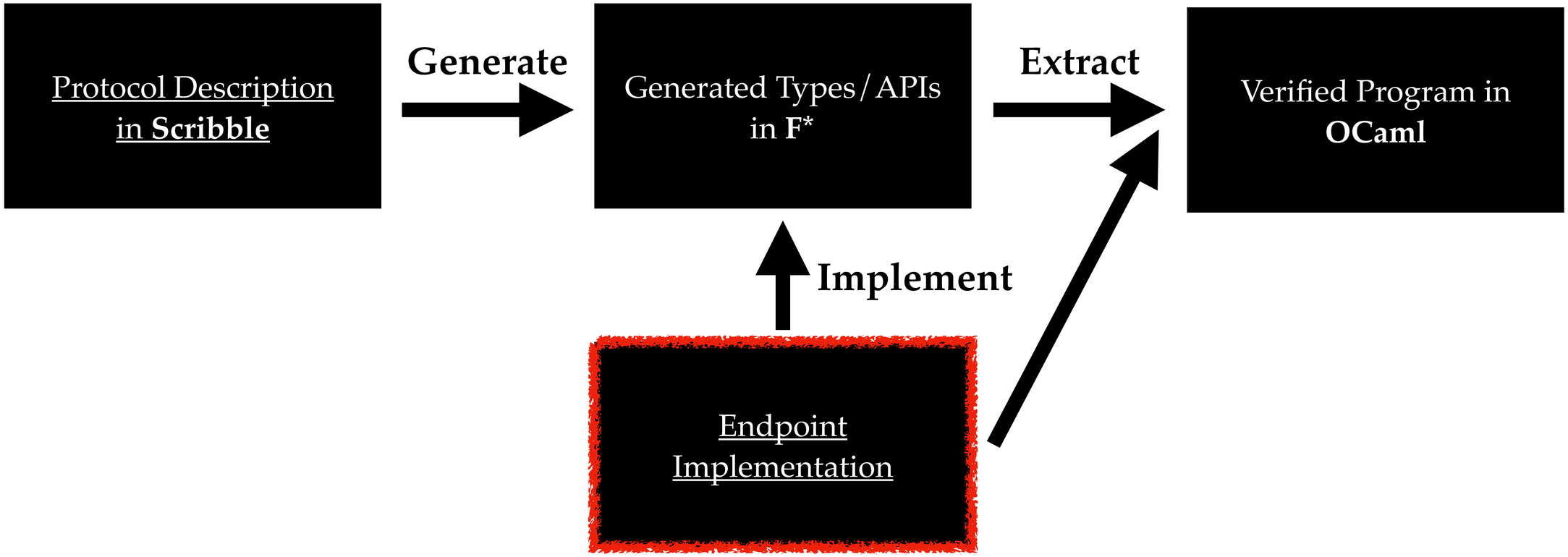
For the external choice of C

```
type receive_Correct = (st: state)
    → unit{st.g = st.s} → unit
type receive_Higher = (st: state)
    → unit{st.g < st.s} → unit
type receive_Lower = (st: state)
    → unit{st.g > st.s} → unit
```

`st.s` carries an erased type.

Session* - A Toolchain for RMPST

Workflow



Implementing Endpoints

Generated API

For the internal choice of B

```
type choice (st: state) =
| Correct of unit{st.g = st.s}
| Higher of unit{st.g < st.s}
| Lower of unit{st.g > st.s}

type choice_send = (st: state) → choice st
```

Implementation

```
let send_handler (st: state) =
  if st.g = st.s then Correct ()
  else if st.g < st.s then Higher ()
                      else Lower ()
```

Implementing Endpoints

Generated API

For the internal choice of B

```
type choice (st: state) =
| Correct of unit{st.g = st.s}
| Higher of unit{st.g < st.s}
| Lower of unit{st.g > st.s}

type choice_send = (st: state) → choice st
```

Implementation

```
let send_handler (st: state) =
  if st.g = st.s then Correct ()
  else if st.g < st.s then Higher ()
                      else Lower ()

let send_handler_err (st: state) =
  if st.g = st.s then Correct ()
  else if st.g > st.s then Higher ()
                      else Lower ()
```

Implementing Endpoints

Generated API

For the internal choice of B

```
type choice (st: state) =
| Correct of unit{st.g = st.s}
| Higher of unit{st.g < st.s}
| Lower of unit{st.g > st.s}

type choice_send = (st: state) → choice st
```

Implementation

```
let send_handler (st: state) =
  if st.g = st.s then Correct ()
  else if st.g < st.s then Higher ()
                      else Lower ()

let send_handler_err (st: state) =
  if st.g = st.s then Correct ()
  else if st.g > st.s then Higher ()
                      else Lower ()
```

Type Error detected statically

Future Work

- Study how to mix verified and non-verified participants
  - To overcome the difficulty that refinement types are not yet prevalent
  - Widens the applicability of our approach
- Improve the expressiveness of the toolchain
  - e.g. Custom-specified predicates

Thank you!