Fluid Types

Statically Verified Distributed Protocols with Refinements

Fangyi Zhou Francisco Ferreira
Rumyana Neykova Nobuko Yoshida

PLACES 2019

Imperial College London

Brunel University London

Example: a simple protocol

- Two kids are playing a game on the playground
- **A** tells **B** a number
- **B** tries to find a larger number

```
protocol Playground (role A, role B) {
    initialGuess (int) from A to B;
    finalGuess (int) from B to A;
}
```

No guarantee whether this will be larger

Example: a simple protocol

- Two kids are playing a game on the playground
- **A** tells **B** a number
- **B** tries to find a larger number

```
protocol Playground (role A, role B) {
    initialGuess (x:int) from A to B @ x > 7;
    finalGuess (y:int) from B to A @ y > x;
}
```

Named Parameters **Assertions**

Previously…

- Session Type Provider [Neykova et al. 2018]
  - Compile Time Type Generation in F#
  - Protocol validated during compilation
  - Refinements checked dynamically during execution

[Neykova et al. 2018]: Rumyana Neykova, Raymond Hu, Nobuko Yoshida, and Fahd Abdeljallal. 2018. A session type provider: compile-time API generation of distributed protocols with refinements in F#

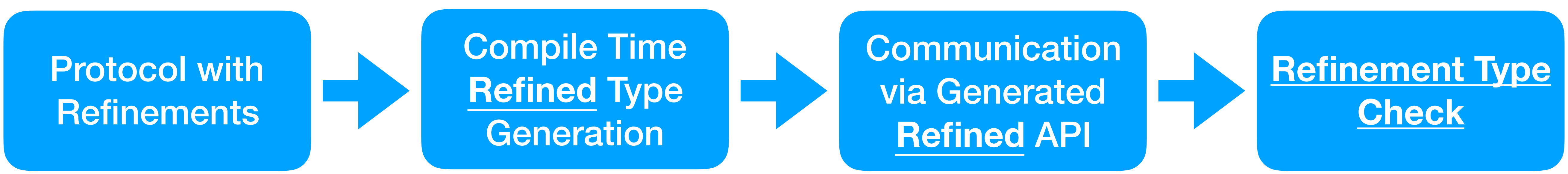
Workflow (Previously)


```
protocol Playground
    (role A, role B) {
  initialGuess (x:int)
    from A to B @ x > 7;
  finalGuess (y:int)
    from B to A @ y > x;
}
```

```
type Protocol
  = SessionTypeProvider
    <“Playground.scr”, “A”>
```

```
let p =
  new Protocol().Init()
in
  p.send(B, initialGuess, 42)
   .receive(B, finalGuess, y)
   .finish()
```

Workflow (Now)



Overview

- Add refinements to generated types
- Check refinements with a type system extension
    - Extract F# code into a refinement calculus
    - Check satisfiability using external solver

What are refinement types?

- Build upon an existing type system
- Allow base types to be refined via predicates
- Specify data dependencies
- Example: Liquid Haskell [Vazou et al. 2014]

[Vazou et al. 2014]: Niki Vazou, Eric L. Seidel, Ranjit Jhala, Dimitrios Vytiniotis, and Simon Peyton-Jones. 2014. Refinement types for Haskell.

Refinement Calculus: λ^H

- STLC with refinement types
- Terms can be encoded in SMT-LIB terms
- Establishes a subtyping relation via SMT solver

Types in λ^H

integers, booleans, …

- A base type

$$\{\nu : b \mid M\}$$

Base type b, value ν refined by term M

- A function type (dependent function)

$$(x : \tau_1) \to \tau_2$$

Variable x can occur in the type τ_2

c.f. Dependent Types $\prod_{x:\tau_1} \tau_2(x)$

Example

- The integer literal `1`
  - A possible type: $\{\nu : \textbf{int} \mid \nu = 1\}$
  - Another possible type: $\{\nu : \textbf{int} \mid \nu \geq 1\}$
  - Or more… $\{\nu : \textbf{int} \mid \textbf{true}\}$
- Solution: Bidirectional Typing

Bidirectional Typing

- Provides a more algorithmic approach
- Mutually inductive judgments
- Type Synthesis

$$\Gamma; \Delta \vdash M \overset{*}{\Rightarrow} \tau$$

Given Γ, Δ, M, find the type τ

***Not all terms are synthesisable**

- Type Check

$$\Gamma; \Delta \vdash M \Leftarrow \tau$$

Given Γ, Δ, M, τ, determine if type is correct

“Change of Direction” Rule

Subtyping Judgment **Well-formedness Judgment**

$$\frac{\Gamma;\Delta \vdash \tau <: \tau' \qquad \Gamma;\Delta \vdash M \Rightarrow \tau \qquad \Gamma;\Delta \vdash \tau'}{\Gamma;\Delta \vdash M \Leftarrow \tau'}$$

Subtyping with SMT

- Encode refinements term into SMT-LIB
- Use SMT solver to decide validity

$$\frac{\mathsf{Valid}(\llbracket \Gamma \rrbracket \wedge \llbracket \Delta \rrbracket \wedge \llbracket M_1 \rrbracket \implies \llbracket M_2 \rrbracket)}{\Gamma, \Delta \vdash \{v : b \mid M_1\} <: \{v : b \mid M_2\}}$$

Encoding in SMT-LIB

x **(A term Variable)** → x **(An SMT Variable)**

Encoding in SMT-LIB

$(+)\ 1\ 2 \longrightarrow (+\ 1\ 2)$

Encoding in SMT-LIB

$x : \{\nu : \textbf{int} \mid \nu + 2 = 5\}$ $\longrightarrow$ $x + 2 = 5$

Encoding in SMT-LIB

Valid$(\llbracket \Gamma \rrbracket \wedge \llbracket \Delta \rrbracket \wedge \llbracket M_1 \rrbracket \implies \llbracket M_2 \rrbracket)$

Unsat$(\llbracket \Gamma \rrbracket \wedge \llbracket \Delta \rrbracket \wedge \llbracket M_1 \rrbracket \wedge \neg \llbracket M_2 \rrbracket)$

Subtyping with SMT

- Consider integer literal `1`
  - Synthesised type: $\{\nu : \textbf{int} \mid \nu = 1\}$
  - Check subtype: $\{\nu : \textbf{int} \mid \nu = 1\} <: \{\nu : \textbf{int} \mid \nu \geq 1\}$?
  - Encode into logic: $\textbf{SAT}((v = 1) \wedge \neg(v \geq 1))$?
  - Use SMT solver: **UNSAT**

Subtyping with SMT

- Consider term `x + 1` with context $x : \{\nu : \textbf{int} \mid \nu \geq 1\}$
  - Synthesised type: $\{\nu : \textbf{int} \mid \nu = x + 1\}$
  - Check subtype: $\{\nu : \textbf{int} \mid \nu = x + 1\} <: \{\nu : \textbf{int} \mid \nu \geq 2\}?$
  - Encode into logic: $\textbf{SAT}((x \geq 1) \wedge (v = x + 1) \wedge \neg(v \geq 2))?$
  - Use SMT solver: **UNSAT**

Generating Types

- Scribble validates protocol and generates CFSM
- Type Provider converts CFSM into F# code
- New: Adding refinements in types

From Protocol to CFSM (Scribble)

```
protocol Playground (role A, role B) {
    initialGuess (x:int) from A to B @ x > 7;
    finalGuess (y:int) from B to A @ y > x;
}
```

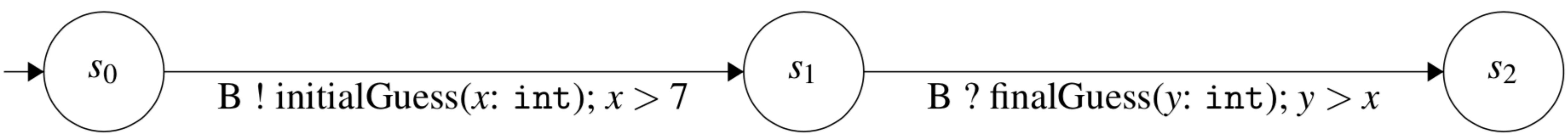
Projection to role A

protocol Playground (role A, role B) {
    initialGuess (x:int) ~~from A~~ to B @ x > 7;
    finalGuess (y:int) from B ~~to A~~ @ y > x;
}

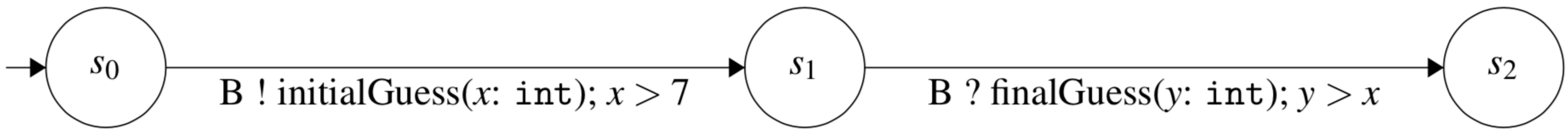
From Protocol to CFSM (Scribble)

```
protocol Playground (role A, role B) {
    initialGuess (x:int) from A to B @ x > 7;
    finalGuess (y:int) from B to A @ y > x;
}
```

Projection to role `A`



From CFSM to λ^H (Type Provider)



$\emptyset$

$x : \{\nu : \textbf{int} \mid \nu > 7\}$

$x : \{\nu : \textbf{int} \mid \nu > 7\}$

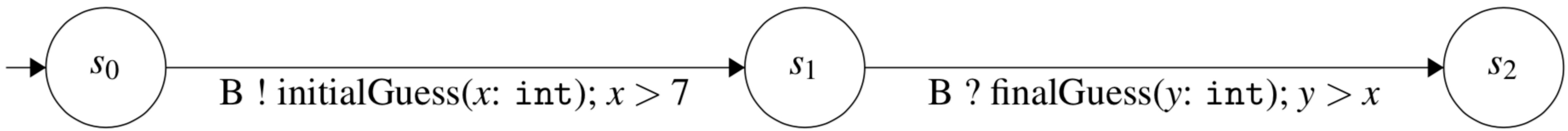
$y : \{\nu : \textbf{int} \mid \nu > x\}$

```
type State0 = {}
```

```
type State1 = {
  x: {v:int|v>7};
}
```

```
type State2 = {
  x: {v:int|v>7};
  y: {v:int|v>x};
}
```

From CFSM to λ^H
(Type Provider)



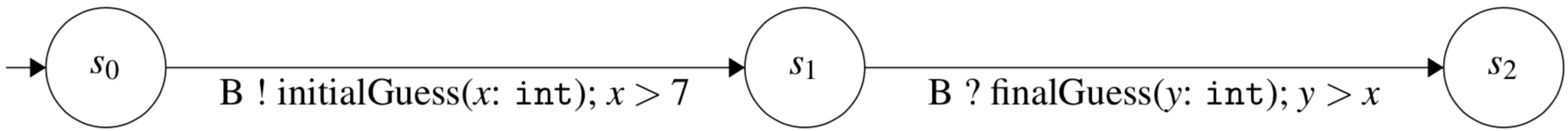

```
type State0 = {}
```

```
type State1 = {
  x: {v:int|v>7};
}
```

```
type State2 = {
  x: {v:int|v>7};
  y: {v:int|v>x};
}
```

```
initialGuess : (st: State0) -> (x: {v:int|v>7}) -> State1
```

From CFSM to λ^H
(Type Provider)



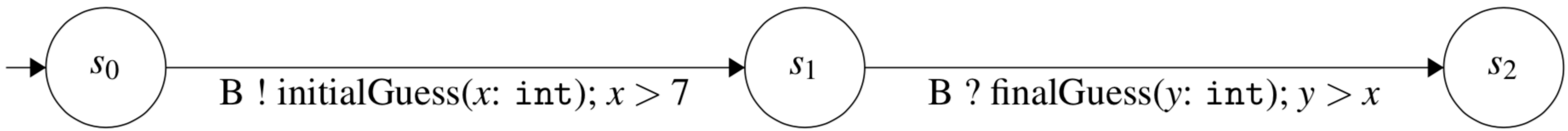

```
type State0 = {}
```

```
type State1 = {
  x: {v:int|v>7};
}
```

```
type State2 = {
  x: {v:int|v>7};
  y: {v:int|v>x};
}
```

```
finalGuess : (st: State1) -> (State2 * {v:int|v>st.x})
```

From CFSM to λ^H (Type Provider)




```
type State0 = {}
```

```
type State1 = {
  x: {v:int|v>7};
}
```

```
type State2 = {
  x: {v:int|v>7};
  y: {v:int|v>x};
}
```

```
initialGuess : (st: State0) -> (x: {v:int|v>7}) -> State1
  finalGuess : (st: State1) -> (State2 * {v:int|v>st.x})
```

One Last Step…

- Typecheck the program with refined types
  - Extract F# expressions to terms in λ^H
  - Use F# Compiler Services to obtain AST
  - Check whether API usage is correct w.r.t. refinements

Future Work

- Support recursion in protocols
- Complete meta-theory for refinements in MPST
  - End to end meta-theory
- Support more features in refinement calculus

Thank you!