# Type-safe Bidirectional Channels in Idris 2

Guillaume Allais

University of Strathclyde
Glasgow, UK

June 12$^{th}$ 2025

TYPES

# Table of Contents

Protocols

Protocol-respecting Programs

Typed Communications

Stateful Protocols

# Table of Contents

Protocols

Protocol-respecting Programs

Typed Communications

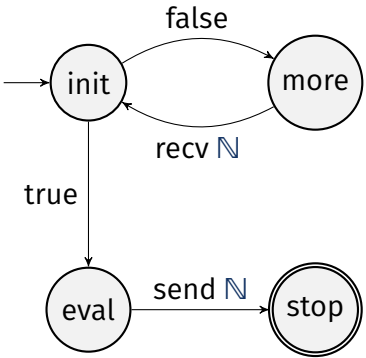
Stateful Protocols

# Session types

$$
\begin{aligned}
p,\ q,\ \ldots\ ::= {} & x \mid \mu x.\ p \mid \nu x.\ p \\
& \mid\ !A.\ p \mid\ ?A.\ p \mid \texttt{end} \\
& \mid p \,\&\, q \mid p \oplus q
\end{aligned}
$$

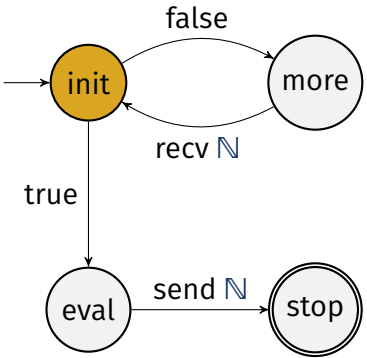
# The “adder” Protocol

$$\nu i.\ ((!\mathbb{N}.\ \mathtt{end})\,\&\,(?\mathbb{N}.\ i))$$



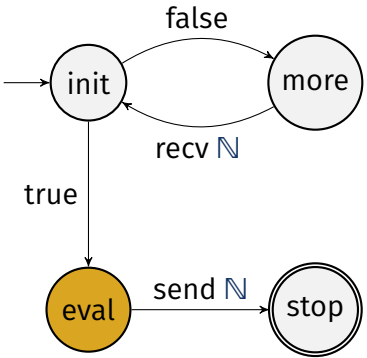
# The “adder” Protocol

$\nu i.\ ((!\mathbb{N}.\ \mathtt{end})\,\&\,(?\mathbb{N}.\ i))$



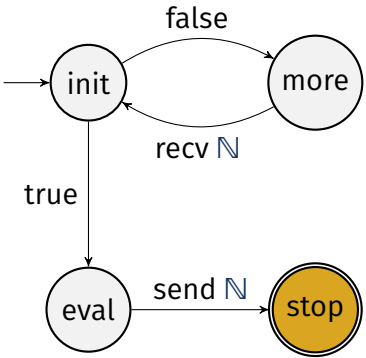
# The “adder” Protocol

$$\nu i.\ (\ (!\mathbb{N}.\ \mathtt{end})\ \&(?\mathbb{N}.\ i))$$



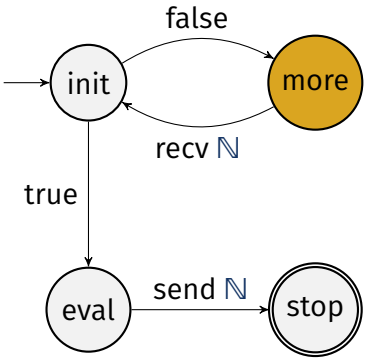
# The “adder” Protocol

$$\nu i.\ ((!\mathbb{N}.\ \texttt{end})\ \&\ (?\mathbb{N}.\ i))$$



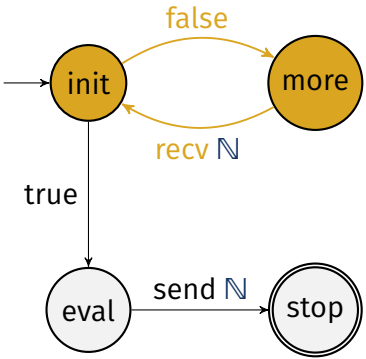
# The “adder” Protocol

$$\nu i.\ ((!\mathbb{N}.\ \mathtt{end})\ \&\ (?\mathbb{N}.\ i))$$



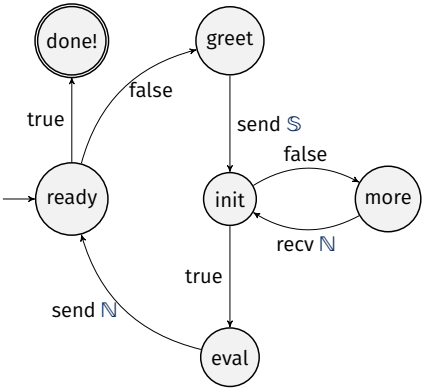
# The “adder” Protocol

$$\nu\, i.\ ((!\mathbb{N}.\ \mathtt{end})\ \&\ (?\mathbb{N}.\ i))$$



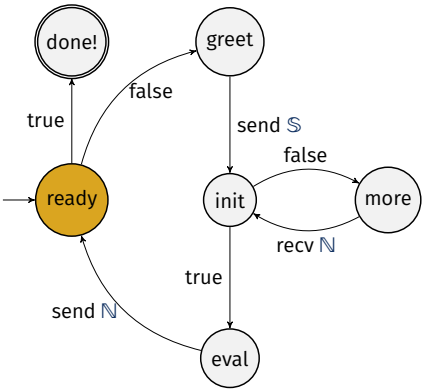
# The “server” Protocol

$$\nu r.\ (\mathtt{end} \,\&\, (!\mathbb{S}.\ \nu i.\ ((!\mathbb{N}.\ r) \,\&\, (?\mathbb{N}.\ i))))$$



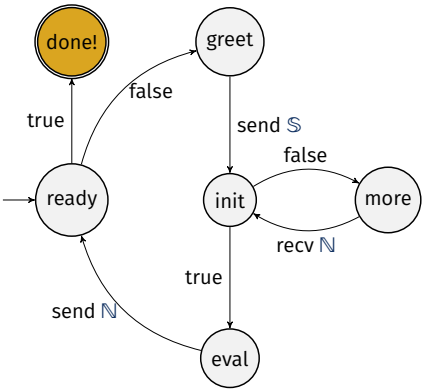
# The “server” Protocol

$$\nu r.\ (\texttt{end} \,\&\, (!\mathbb{S}.\ \nu i.\ ((!\mathbb{N}.\ r) \,\&\, (?\mathbb{N}.\ i))))$$



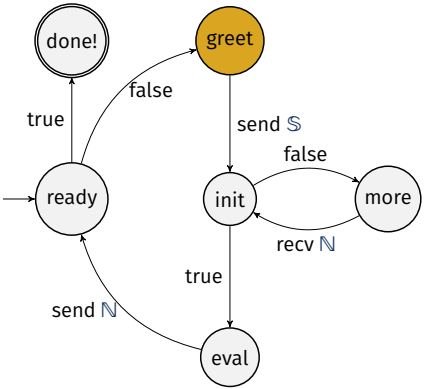
# The “server” Protocol

$$\nu r.\ (\ \texttt{end}\ \&(!\mathbb{S}.\ \nu i.\ ((!\mathbb{N}.\ r)\,\&(?\mathbb{N}.\ i))))$$



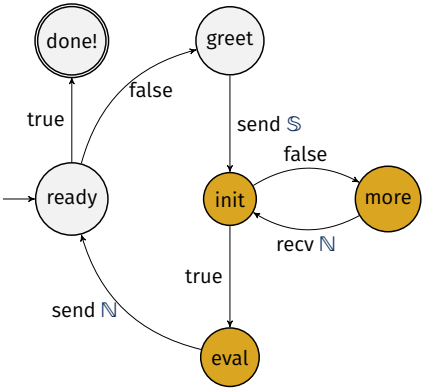
# The “server” Protocol

$$\nu r.\,(\texttt{end}\,\&\,(!\mathbb{S}.\,\nu i.\,((!\mathbb{N}.\,r)\,\&\,(?\mathbb{N}.\,i))))$$



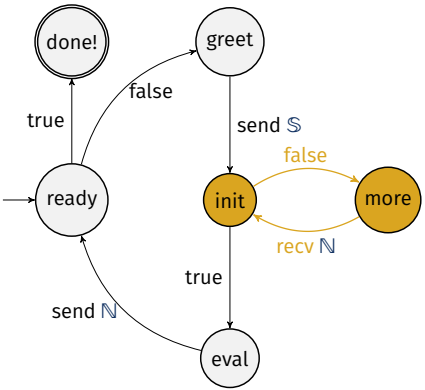
# The “server” Protocol

$$\nu r.\ (\mathtt{end} \,\&\, (!\mathbb{S}.\ \nu i.\ ((!\mathbb{N}.\ r) \,\&\, (?\mathbb{N}.\ i))))$$



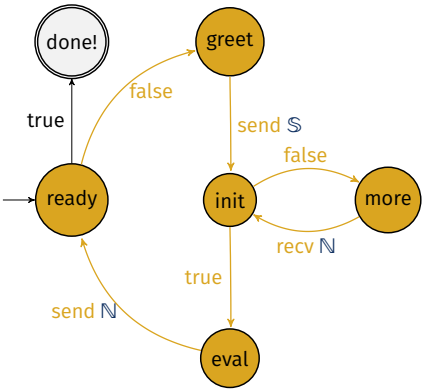
# The “server” Protocol

$$\nu r.\ (\mathtt{end} \,\&\, (!\mathbb{S}.\ \nu\, i.\ ((!\mathbb{N}.\ r) \,\&\, (?\mathbb{N}.\ i))))$$



# The “server” Protocol

$$\nu\, r.\ (\mathtt{end} \,\&\, (!\mathbb{S}.\ \nu i.\ ((!\mathbb{N}.\ r) \,\&\, (?\mathbb{N}.\ i))))$$



# Table of Contents

Protocols

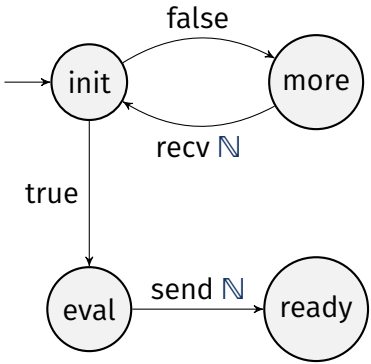
Protocol-respecting Programs

Typed Communications

Stateful Protocols

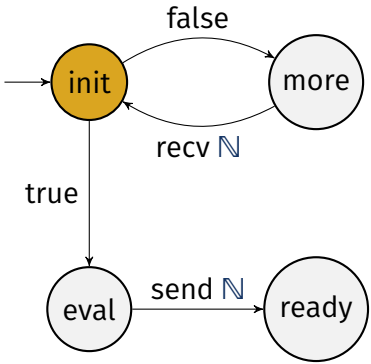
# The “adder” Implementation

```
adder acc ch = do
  ch <- unroll {nm = Nu "adder"} ch
  b # ch <- offer ch
  if b then do
      ch <- send ch acc
      rec {nm = Nu "ready"} ch
    else do
      (n # ch) <- recv ch
      ch <- rec {nm = Nu "adder"} ch
      adder (acc + n) ch
```



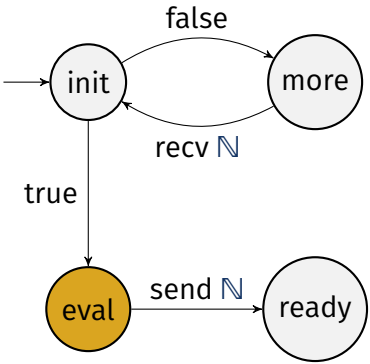
# The “adder” Implementation

```
adder acc ch = do
  ch <- unroll {nm = Nu "adder"} ch
  b # ch <- offer ch
  if b then do
      ch <- send ch acc
      rec {nm = Nu "ready"} ch
    else do
      (n # ch) <- recv ch
      ch <- rec {nm = Nu "adder"} ch
      adder (acc + n) ch
```



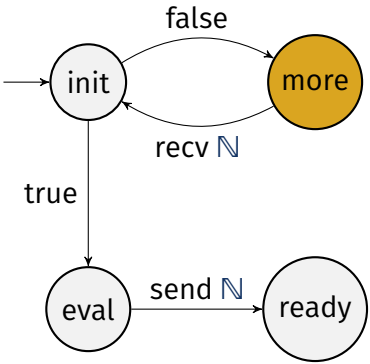
# The “adder” Implementation

```
adder acc ch = do
  ch <- unroll {nm = Nu "adder"} ch
  b # ch <- offer ch
  if b then do
     ch <- send ch acc
     rec {nm = Nu "ready"} ch
   else do
     (n # ch) <- recv ch
     ch <- rec {nm = Nu "adder"} ch
     adder (acc + n) ch
```



# The “adder” Implementation

```
adder acc ch = do
  ch <- unroll {nm = Nu "adder"} ch
  b # ch <- offer ch
  if b then do
      ch <- send ch acc
      rec {nm = Nu "ready"} ch
    else do
      (n # ch) <- recv ch
      ch <- rec {nm = Nu "adder"} ch
      adder (acc + n) ch
```



# Table of Contents

Protocols

Protocol-respecting Programs

Typed Communications

Stateful Protocols

# Idris' Typed Channels

```
data Channel : Type -> Type where [external]

channelGet : HasIO io => Channel a -> io a

channelPut : HasIO io => Channel a -> a -> io ()
```

# Idris' Typed Channels

```
data Channel : Type -> Type where [external]

channelGet : HasIO io => Channel a -> io a

channelPut : HasIO io => Channel a -> a -> io ()
```

- Pick one type and stick with it

# Idris' Typed Channels

```
data Channel : Type -> Type where [external]

channelGet : HasIO io => Channel a -> io a

channelPut : HasIO io => Channel a -> a -> io ()
```

- Pick one type and stick with it
- No protocol

# Messages Type

A big (and cheap) union of types

```
data UnionT : (elt : a -> Type) -> (ts : List a) -> Type where
  Element : (k : Nat) -> (0 _ : AtIndex t ts k) -> elt t -> UnionT elt ts
```

# Messages Type

A big (and cheap) union of types

```
data UnionT : (elt : a -> Type) -> (ts : List a) -> Type where
  Element : (k : Nat) -> (0 _ : AtIndex t ts k) -> elt t -> UnionT elt ts
```

Listing all of the sent and received types

```
SendTypes : Session nms m -> List Type
RecvTypes : Session nms m -> List Type
```

# Table of Contents

Protocols

Protocol-respecting Programs

Typed Communications

Stateful Protocols

# Stateful Protocols

Communications provoke destructive updates of the channels

# Stateful Protocols

Communications provoke destructive updates of the channels

- (Ab)using linearity to encode uniqueness

# Stateful Protocols

Communications provoke destructive updates of the channels

- (Ab)using linearity to encode uniqueness
- **Not** unfolding fixpoints by substitution

# Stateful Protocols

Communications provoke destructive updates of the channels

- (Ab)using linearity to encode uniqueness
- **Not** unfolding fixpoints by substitution
- Recording where we are in the state machine

# Syntax of Session Types

```
data Kind : Type where
  Mu, Nu : Name -> Kind
```

```
data Session : List Kind -> Norm -> Type where
  Fix : (kd : Kind) -> Session (kd :: nms) Head -> Session nms Head
  Rec : {kd : Kind} -> Focus kd nms -> Session nms Expr
```

# Syntax of Session Types

```
data Kind : Type where
  Mu, Nu : Name -> Kind
```

```
data Session : List Kind -> Norm -> Type where
  Fix : (kd : Kind) -> Session (kd :: nms) Head -> Session nms Head
  Rec : {kd : Kind} -> Focus kd nms -> Session nms Expr
  Send : (ty : Type) -> Session nms n -> Session nms Head
  Recv : (ty : Type) -> Session nms n -> Session nms Head
  End : Session nms Head
  Offer : Session nms m1 -> Session nms m2 -> Session nms Head
  Select : Session nms m1 -> Session nms m2 -> Session nms Head
```

# Environment Stack

To make sense of nested fixpoints, we need a telescopic environment

```
data Env : List Kind -> Type where
  Nil  : Env []
  (::) : Session (nm :: nms) Head -> Env nms -> Env (nm :: nms)
```

# Channels

```
record Channel (s : Session nms k) (e : Env nms) where
  constructor MkChannel
  {sendStep    : Nat}
  {recvStep    : Nat}
  {0 ogNorm    : Norm}
  {0 ogKinds   : Kinds}
  {0 ogSession : Session ogKinds ogNorm}
  context      : Context recvStep sendStep ogSession s e

  sendChan : Threads.Channel (Union (Bool :: SendTypes ogSession))
  recvChan : Threads.Channel (Union (Bool :: RecvTypes ogSession))
```

# Channels

```
record Channel (s : Session nms k) (e : Env nms) where
  constructor MkChannel
  {sendStep    : Nat}
  {recvStep    : Nat}
  {0 ogNorm    : Norm}
  {0 ogKinds   : Kinds}
  {0 ogSession : Session ogKinds ogNorm}
  context      : Context recvStep sendStep ogSession s e

  sendChan : Threads.Channel (Union (Bool :: SendTypes ogSession))
  recvChan : Threads.Channel (Union (Bool :: RecvTypes ogSession))
```

# Basic Building Blocks

```
send : Channel (Send ty s) e -@ (ty -> IO1 (Channel s e))
recv : Channel (Recv ty s) e -@ IO1 (Res ty (\ _ => Channel s e))
```

# Basic Building Blocks

```
send : Channel (Send ty s) e -@ (ty -> IO1 (Channel s e))
recv : Channel (Recv ty s) e -@ IO1 (Res ty (\ _ => Channel s e))
```

```
offer : Channel (Offer s1 s2) e -@
  IO1 (Res Bool (\ b => ifThenElse b (Channel s1 e) (Channel s2 e)))
select : Channel (Select s1 s2) e -@
  ((b : Bool) -> IO1 (ifThenElse b (Channel s1 e) (Channel s2 e)))
```

# Basic Building Blocks

```
send : Channel (Send ty s) e -@ (ty -> IO1 (Channel s e))
recv : Channel (Recv ty s) e -@ IO1 (Res ty (\ _ => Channel s e))
```

```
offer : Channel (Offer s1 s2) e -@
  IO1 (Res Bool (\ b => ifThenElse b (Channel s1 e) (Channel s2 e)))
select : Channel (Select s1 s2) e -@
  ((b : Bool) -> IO1 (ifThenElse b (Channel s1 e) (Channel s2 e)))
```

```
unroll : Channel (Fix nm s) e -@ IO1 (Channel s (s :: e))
roll   : Channel s (s :: e) -@ IO1 (Channel (Fix nm s) e)
```

# Basic Building Blocks

```
send : Channel (Send ty s) e -@ (ty -> IO1 (Channel s e))
recv : Channel (Recv ty s) e -@ IO1 (Res ty (\ _ => Channel s e))
```

```
offer : Channel (Offer s1 s2) e -@
  IO1 (Res Bool (\ b => ifThenElse b (Channel s1 e) (Channel s2 e)))
select : Channel (Select s1 s2) e -@
  ((b : Bool) -> IO1 (ifThenElse b (Channel s1 e) (Channel s2 e)))
```

```
unroll : Channel (Fix nm s) e -@ IO1 (Channel s (s :: e))
roll   : Channel s (s :: e) -@ IO1 (Channel (Fix nm s) e)
```

```
rec : {pos : _} ->  Channel (Rec pos) e -@
  IO1 (Channel (Fix nm (SessionAt pos e)) (EnvAt pos e))
```

# What's next?

- Small runtime overhead
- Ergonomics
- Totality Checking