Fun with Label-Dependent Session Types

Peter Thiemann
Vasco T. Vasconcelos

University of Freiburg
University of Lisbon

IFIFP WG 2.8 meeting
Bordeaux, May 2019

The good old math server

Session type

```
type Server = &{
  Neg: ?Int. !Int. end! ,
  Add: ?Int. ?Int. !Int. end! }
```

The good old math server

Session type

```
type Server = &{
  Neg: ?Int. !Int. end! ,
  Add: ?Int. ?Int. !Int. end! }
```

Implementation

```
server : Server → Unit
Server c =
  rcase c of
    Neg → c. let x, c = recv c
                 c = send c (−x) in
             close c
    Add → c. let x, c = recv c
                 y, c = recv c
                    c = send c (x + y) in
             close c
```

. . . and a client

```
negClient : dualof Server → Int
negClient d x =
  let d = select Neg d
      d = send d x
    r, d = recv d
      _ = wait d in
  r
```

Observation

The I/O nature of channel operations

Output	Input
send c l	**recv** c
select c l	**rcase** c **of** $\{$l1 →c.e1, l2→c.e2$\}$
close c	**wait** c

Observation

The I/O nature of channel operations

Output	Input
send c l	**recv** c
select c l	**rcase** c **of** {l1 →c.e1, l2→c.e2}
close c	**wait** c

Scope for unification

First-class labels

$$
\begin{array}{lcl}
\textbf{select}\ c\ l & \rightsquigarrow & \textbf{send}\ c\ l \\
\textbf{rcase}\ c\ \textbf{of}\ \{l1 \to c.e1,\ l2 \to c.e2\} & \rightsquigarrow & \\
\qquad \textbf{let}\ c, l = \textbf{recv}\ c\ \textbf{in} & \textbf{case} & l\ \textbf{of}\ \{l1 \to e1,\ l2 \to e2\} \\
\textbf{close}\ c & \rightsquigarrow & \textbf{send}\ c\ \mathrm{EOS} \\
\textbf{wait}\ c & \rightsquigarrow & \textbf{recv}\ c
\end{array}
$$

The pre-syntax of types

$$(x{:}\,A) \to B \qquad \{l_1, \ldots, l_n\}$$

$$(x{:}\,A) \times B \qquad \textbf{case } V \textbf{ of } \{l_i \to A_i\}$$

$$(x{:}\,A)\ !\ B \qquad \textbf{Unit}$$

$$(x{:}\,A)\ ?\ B \qquad V = W$$

The label-dependent math server

```
type LServer =
  (l: {Neg, Add}) ? case l of
    Neg → Int?Int!{EOS}!Unit
    Add → Int?Int?Int!{EOS}!Unit
```

The label-dependent math server

```
type LServer =
  (l: {Neg, Add}) ? case l of
    Neg → Int?Int!{EOS}!Unit
    Add → Int?Int?Int!{EOS}!Unit

lServer : LServer → Unit
lServer c =
  let l, c = recv c
  in case l of
    Neg → let x, c = recv c in
          send (send c (−x)) EOS
    Add → let x, c = recv c
              y, c = recv c in
          send (send c (x+y)) EOS
```

Advantages

- Smaller unified operational semantics
- More flexibility to implement a type
- Immediate correspondences on type level

Taming the beast

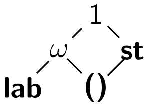
- Linearity
- Dependency

Multiplicities

multiplicities	m	$::= \; 1 \mid \omega$
occurrences	o	$::= \; 0 \mid m$
environments	Γ	$::= \; \cdot \mid \Gamma, x :^{o} A$
kinds	K	$::= \; m \mid \mathbf{lab} \mid \mathbf{st} \mid \mathbf{()}$

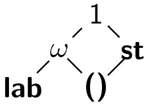
Multiplicities

multiplicities	m	$::=$	$1 \mid \omega$
occurrences	o	$::=$	$0 \mid m$
environments	Γ	$::=$	$\cdot \mid \Gamma, x :^{o} A$
kinds	K	$::=$	$m \mid \mathbf{lab} \mid \mathbf{st} \mid \mathbf{()}$



Multiplicities

$$
\begin{array}{rrcl}
\text{multiplicities} & m & ::= & 1 \mid \omega \\
\text{occurrences} & o & ::= & 0 \mid m \\
\text{environments} & \Gamma & ::= & \cdot \mid \Gamma, x :^{o} A \\
\text{kinds} & K & ::= & m \mid \mathbf{lab} \mid \mathbf{st} \mid \mathbf{()}
\end{array}
$$



demotion $\quad \downarrow 0 = 0 \quad \downarrow 1 = 0 \quad \downarrow \omega = \omega$

Output formation & elimination

$$\frac{\downarrow\Gamma \vdash A : m \qquad \Gamma, x :^{\downarrow m} A \vdash B : \mathbf{st}}{\Gamma \vdash (x : A)!B : \mathbf{st}}$$

$$\frac{\Gamma \vdash M : (x : A)!B}{\Gamma \vdash \mathbf{send}\, M : (x : A) \to B}$$

Input formation & elimination

$$\frac{\downarrow\Gamma \vdash A : m \qquad \Gamma, x :^{\downarrow m} A \vdash B : \mathbf{st}}{\Gamma \vdash (x : A)?B : \mathbf{st}}$$

$$\frac{\Gamma \vdash M : (y : A)?B}{\Gamma \vdash \mathbf{recv}\, M : (y : A) \times B}$$

Label formation & introduction

$$\frac{\vdash \Gamma : \omega}{\Gamma \vdash L : \mathbf{lab}}$$

$$\frac{\vdash \Gamma : \omega \qquad l \in L}{\Gamma \vdash l : L}$$

L is a non-empty set of labels

Case formation & case introduction; label elimination

$$\frac{\downarrow\Gamma \vdash V : \{\overline{l_i}\} \qquad \Gamma, \_ :^{\omega} V = l_i \vdash A_i : K \quad (\forall i)}{\Gamma \vdash \mathbf{case}\ V\ \mathbf{of}\ \{\overline{l_i \to A_i}\} : K}$$

$$\frac{\downarrow\Gamma \vdash V : \{\overline{l_i}\} \qquad \Gamma, \_ :^{\omega} V = l_i \vdash N_i : A \quad (\forall i)}{\Gamma \vdash \mathbf{case}\ V\ \mathbf{of}\ \{\overline{l_i \to N_i}\} : A}$$

Value equality as a type

$$\frac{\Gamma \vdash V : A \qquad \Gamma \vdash W : A \qquad \Gamma \vdash A : \mathbf{lab}}{\Gamma \vdash V = W : \mathbf{un}}$$

No introduction rules
Eliminated by type conversion
Type $V = W$ inhabited by evidence that values V and W are equal
Introduced in contexts by label elimination

The label-dependent math server, again

```
type LServer =
  (l: {Neg, Add}) ? case l of
    Neg → Int ? Int ! Unit
    Add → Int ? Int ? Int ! Unit
```

The label-dependent math server, again

```
type LServer =
  (l: {Neg, Add}) ? case l of
    Neg → Int ? Int ! Unit
    Add → Int ? Int ? Int ! Unit

lServer : LServer → Unit
lServer c =
  let l, c = recv c in
  case l of
    Neg → let x, c = recv c in
          send c (−x)
    Add → let x, c = recv c
              y, c = recv c in
          send c (x + y)
```

This time we do not explicitly close channels

The LD math server, refactored

```
type L = {Neg, Add}
type LServerR =
  (l:L) ? Int ? case l of
    Neg → Int ! Unit
    Add → Int ? Int ! Unit
```

The LD math server, refactored

```
type L = {Neg, Add}
type LServerR =
  (l:L) ? Int ? case l of
    Neg → Int ! Unit
    Add → Int ? Int ! Unit

lServerR : LServerR → Unit
lServerR c =
  let l, c = recv c
      x, c = recv c in
  case l of
    Neg → send c (−x)
    Add → let y, c = recv c in
          send c (x+y)
```

Commuting conversion of **send**/**recv** over **case**

Can we type lServer against LServerR?

$$c :^1 (l : L)?\mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \ldots\}$$

Can we type lServer against LServerR?

$$c :^1 (l : L)?\mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \rightarrow \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \rightarrow \dots\}$$

let l , c = **recv** c **in**

Can we type lServer against LServerR?

$$c :^1 (l : L)?\mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$$

let l, c = **recv** c **in**

$$l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$$

Can we type lServer against LServerR?

$$c :^1 (l : L)?\textbf{Int}?\textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}!\textbf{Unit}, \text{Add} \to \dots\}$$

let l , c = **recv** c **in**

$$l :^\omega L, c :^1 \textbf{Int}?\textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}!\textbf{Unit}, \text{Add} \to \dots\}$$

case l **of**

Neg →

Can we type lServer against LServerR?

$$c :^1 (l : L)?\mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\text{Neg} \to \mathbf{Int}!\mathbf{Unit}, \text{Add} \to \dots\}$$

let l , c = **recv** c **in**

$$l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\text{Neg} \to \mathbf{Int}!\mathbf{Unit}, \text{Add} \to \dots\}$$

case l **of**

Neg →

$$\_ :^\omega l = \text{Neg}, l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\text{Neg} \to \mathbf{Int}!\mathbf{Unit}, \text{Add} \to \dots\}$$

Can we type lServer against LServerR?

$$c :^1 (l : L)?\mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$$

```
let l, c = recv c in
```

$$l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$$

```
case l of
  Neg →
```

$$\_ :^\omega l = \mathrm{Neg}, l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$$

```
    let x, c = recv c
```

Can we type lServer against LServerR?

$$c :^1 (l : L)?\mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathsf{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathsf{Add} \to \dots\}$$

let l , c = **recv** c **in**

$$l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathsf{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathsf{Add} \to \dots\}$$

case l **of**

Neg →

$$\_ :^\omega l = \mathsf{Neg}, l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathsf{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathsf{Add} \to \dots\}$$

let x , c = **recv** c

$$x :^\omega \mathbf{Int}, \_ :^\omega l = \mathsf{Neg}, l :^\omega L, c :^1 \mathbf{case}\ l\ \mathbf{of}\{\mathsf{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathsf{Add} \to \dots\}$$

Can we type lServer against LServerR?

$$c :^1 (l : L)?\textbf{Int}?\textbf{case } l \textbf{ of}\{\text{Neg} \rightarrow \textbf{Int}!\textbf{Unit}, \text{Add} \rightarrow \ldots\}$$

let l , c = **recv** c **in**

$$l :^\omega L, c :^1 \textbf{Int}?\textbf{case } l \textbf{ of}\{\text{Neg} \rightarrow \textbf{Int}!\textbf{Unit}, \text{Add} \rightarrow \ldots\}$$

case l **of**
  Neg →

$$\_ :^\omega l = \text{Neg}, l :^\omega L, c :^1 \textbf{Int}?\textbf{case } l \textbf{ of}\{\text{Neg} \rightarrow \textbf{Int}!\textbf{Unit}, \text{Add} \rightarrow \ldots\}$$

let x, c = **recv** c

$$x :^\omega \textbf{Int}, \_ :^\omega l = \text{Neg}, l :^\omega L, c :^1 \textbf{case } l \textbf{ of}\{\text{Neg} \rightarrow \textbf{Int}!\textbf{Unit}, \text{Add} \rightarrow \ldots\}$$

send c (−x)   −− *we need c: Int !Unit*

Can we type lServer against LServerR?

$c :^1 (l : L)?\mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$

let l , c = **recv** c **in**

$l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$

case l **of**
Neg →

$\_ :^\omega l = \mathrm{Neg}, l :^\omega L, c :^1 \mathbf{Int}?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$

let x , c = **recv** c

$x :^\omega \mathbf{Int}, \_ :^\omega l = \mathrm{Neg}, l :^\omega L, c :^1 \mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \dots\}$

send c (−x) -- *we need c: Int !Unit* We need: **case** l **of** C ≡ **case** Neg **of** C ≡ **Int** ! **Unit**

Type Equivalence

$$\frac{\Gamma \vdash \_ : V = W}{\Gamma \vdash \textbf{case}\ V\ \textbf{of}\ \{l_i \to A_i\} \equiv \textbf{case}\ W\ \textbf{of}\ \{l_i \to A_i\}}$$

$$\frac{}{\Gamma \vdash \textbf{case}\ l_j\ \textbf{of}\ \{l_i \to A_i\} \equiv A_j}$$

Can we type lServerR against LServer?

$$c :^1 (l : L)?\textbf{case } l \textbf{ of}\{\mathsf{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \mathsf{Add} \to \textbf{Int}?A\}$$

Can we type lServerR against LServer?

$$c :^1 (l : L)?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}?\mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \mathbf{Int}?A\}$$

```
let l, c = recv c
```

Can we type lServerR against LServer?

$$c :^1 (l : L)?\textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \text{Add} \to \textbf{Int}?A\}$$

let l, c = **recv** c

$$l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \text{Add} \to \textbf{Int}?A\}$$

Can we type lServerR against LServer?

$$c :^1 (l : L)?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}?\mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \mathbf{Int}?A\}$$

let l, c = **recv** c

$$l :^\omega L, c :^1 \mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}?\mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \mathbf{Int}?A\}$$

let x, c = **recv** c   -- *we need c : Int?case ...*

Can we type lServerR against LServer?

$$c :^1 (l : L)?\textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \text{Add} \to \textbf{Int}?A\}$$

let l, c = **recv** c

$$l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \text{Add} \to \textbf{Int}?A\}$$

let x, c = **recv** c   *-- we need c : Int?case ...*

$$x :^\omega \textbf{Int}, l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}!\textbf{Unit}, \text{Add} \to A\}$$

Can we type lServerR against LServer?

$$c :^1 (l : L)?\mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}?\mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \mathbf{Int}?A\}$$

```
let l, c = recv c
```

$$l :^\omega L, c :^1 \mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}?\mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to \mathbf{Int}?A\}$$

```
let x, c = recv c      -- we need c : Int?case ...
```

$$x :^\omega \mathbf{Int}, l :^\omega L, c :^1 \mathbf{case}\ l\ \mathbf{of}\{\mathrm{Neg} \to \mathbf{Int}!\mathbf{Unit}, \mathrm{Add} \to A\}$$

```
case l of
  Neg →
```

Can we type lServerR against LServer?

$$c :^1 (l : L)?\textbf{case}\ l\ \textbf{of}\{\mathrm{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \mathrm{Add} \to \textbf{Int}?A\}$$

let l, c = **recv** c

$$l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\mathrm{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \mathrm{Add} \to \textbf{Int}?A\}$$

let x, c = **recv** c   -- *we need c : Int?case ...*

$$x :^\omega \textbf{Int}, l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\mathrm{Neg} \to \textbf{Int}!\textbf{Unit}, \mathrm{Add} \to A\}$$

case l **of**

Neg →

$$\_ :^\omega l = \mathrm{Neg}, x :^\omega \textbf{Int}, l :^\omega L, c :^1 \textbf{Int}!\textbf{Unit}$$

Can we type lServerR against LServer?

$$c :^1 (l : L)?\textbf{case}\ l\ \textbf{of}\{\mathrm{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \mathrm{Add} \to \textbf{Int}?A\}$$

let l, c = **recv** c

$$l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\mathrm{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \mathrm{Add} \to \textbf{Int}?A\}$$

let x, c = **recv** c   -- *we need c : Int?case ...*

$$x :^\omega \textbf{Int}, l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\mathrm{Neg} \to \textbf{Int}!\textbf{Unit}, \mathrm{Add} \to A\}$$

case l **of**

Neg →

$$\_ :^\omega l = \mathrm{Neg}, x :^\omega \textbf{Int}, l :^\omega L, c :^1 \textbf{Int}!\textbf{Unit}$$

send c (−x)

Can we type lServerR against LServer?

$$c :^1 (l : L)?\textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \text{Add} \to \textbf{Int}?A\}$$

let l, c = **recv** c

$$l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}?\textbf{Int}!\textbf{Unit}, \text{Add} \to \textbf{Int}?A\}$$

let x, c = **recv** c   −− *we need c : Int ?case ...*

$$x :^\omega \textbf{Int}, l :^\omega L, c :^1 \textbf{case}\ l\ \textbf{of}\{\text{Neg} \to \textbf{Int}!\textbf{Unit}, \text{Add} \to A\}$$

case l **of**

Neg →

$$\_ :^\omega l = \text{Neg}, x :^\omega \textbf{Int}, l :^\omega L, c :^1 \textbf{Int}!\textbf{Unit}$$

send c (−x)

$$\_ :^\omega l = \text{Neg}, x :^\omega \textbf{Int}, l :^\omega L, c :^\omega \textbf{Unit}$$

Type Equivalence

$$\frac{\Gamma \vdash \_ : V = W}{\Gamma \vdash \textbf{case}\ V\ \textbf{of}\ \{l_i \to A_i\} \equiv \textbf{case}\ W\ \textbf{of}\ \{l_i \to A_i\}}$$

$$\frac{}{\Gamma \vdash \textbf{case}\ l_j\ \textbf{of}\ \{l_i \to A_i\} \equiv A_j}$$

$$\frac{\Gamma \vdash x : L}{\Gamma \vdash A \equiv \textbf{case}\ x\ \textbf{of}\ \{l_i \to A\}}$$

Tagged data & algebraic datatypes

A datatype in Haskell:

```
data Either = Left Int | Right Bool
```

Tagged data & algebraic datatypes

A datatype in Haskell:

```
data Either = Left Int | Right Bool
```

The datatype in label-dependent session types:

```
type Either = (tag:{Left,Right}) x
  case tag of
    Left → Int
    Right → Bool
```

Tagged data & algebraic datatypes

A datatype in Haskell:

```
data Either = Left Int | Right Bool
```

The datatype in label-dependent session types:

```
type Either = (tag:{Left,Right}) x
  case tag of
    Left → Int
    Right → Bool
```

An **Either** channel:

```
type EitherC = (tag: {Left, Right}) !
  case tag of
    Left → Int ! Unit
    Right → Bool ! Unit
```

Tagged data & algebraic datatypes

A datatype in Haskell:

```
data Either = Left Int | Right Bool
```

The datatype in label-dependent session types:

```
type Either = (tag:{Left,Right}) x
  case tag of
    Left → Int
    Right → Bool
```

An **Either** channel:

```
type EitherC = (tag: {Left, Right}) !
  case tag of
    Left → Int ! Unit
    Right → Bool ! Unit
```

Sending an **Either** value on a EitherC channel

```
sendEither : Either → EitherC → Unit
sendEither e c =
  let tag, v = e in send (send c tag) v
```

Typing sendEither

$$m :^{\omega} \text{Either}, c :^{1} \text{EitherC}$$

Typing sendEither

$$m :^{\omega} \text{Either}, c :^{1} \text{EitherC}$$

```
let tag, v = m in
```

Typing sendEither

$$m :^{\omega} \mathsf{Either}, c :^{1} \mathsf{EitherC}$$

```
let tag, v = m in
```

$$v :^{\omega} \mathbf{case}\, \mathsf{tag}\, \mathbf{of}\, \{\dots\}, \mathsf{tag} :^{\omega} \{\mathsf{Left}, \mathsf{Right}\}, m :^{\omega} \mathsf{Either}, c :^{1} \mathsf{EitherC}$$

Typing sendEither

$$m :^{\omega} \mathsf{Either}, c :^{1} \mathsf{EitherC}$$

let tag , v = m **in**

$$v :^{\omega} \textbf{case}\, \mathsf{tag}\, \textbf{of}\, \{\dots\}, \mathsf{tag} :^{\omega} \{\mathsf{Left}, \mathsf{Right}\}, m :^{\omega} \mathsf{Either}, c :^{1} \mathsf{EitherC}$$

let c = **send** c tag

Typing sendEither

$$m :^{\omega} \mathsf{Either}, c :^{1} \mathsf{EitherC}$$

```
let tag, v = m in
```

$$v :^{\omega} \textbf{case}\,\mathsf{tag}\,\textbf{of}\,\{\dots\}, \mathsf{tag} :^{\omega} \{\mathsf{Left}, \mathsf{Right}\}, m :^{\omega} \mathsf{Either}, c :^{1} \mathsf{EitherC}$$

```
let c = send c tag
```

$$\dots, v :^{\omega} \textbf{case}\,\mathsf{tag}\,\textbf{of}\,\{\mathsf{Left} \to \textbf{Int}, \dots\}, c :^{1} \textbf{case}\,\mathsf{tag}\,\textbf{of}\,\{\mathsf{Left} \to \textbf{Int}!\mathit{Unit}, \dots$$

```
let c = send c v -- we need c: Int!Unit,  v: Int
                 --      and c: Bool!Unit, v: Bool
```

Typing sendEither

$$m :^{\omega} \text{Either}, c :^{1} \text{EitherC}$$

```
let tag, v = m in
```

$$v :^{\omega} \textbf{case}\,\text{tag}\,\textbf{of}\,\{\ldots\}, \text{tag} :^{\omega} \{\text{Left}, \text{Right}\}, m :^{\omega} \text{Either}, c :^{1} \text{EitherC}$$

```
let c = send c tag
```

$$\ldots, v :^{\omega} \textbf{case}\,\text{tag}\,\textbf{of}\,\{\text{Left} \to \textbf{Int}, \ldots\}, c :^{1} \textbf{case}\,\text{tag}\,\textbf{of}\,\{\text{Left} \to \textbf{Int}!\mathit{Unit}, \ldots$$

```
let c = send c v -- we need c: Int!Unit,  v: Int
                 --      and c: Bool!Unit, v: Bool
```

$$\ldots, v :^{\omega} \textbf{Int}, c :^{\omega} \textbf{Unit}$$

Following all branches in parallel

when eliminating a Σ type on labels

$$\frac{\begin{array}{cc} \Gamma = \Gamma_1 \curlyvee \Gamma_2 & \Gamma_1 \vdash M : \Sigma_m(x : \{l_i\})B \\ x \in fv(B) & \downarrow\Gamma, x :^{\omega} \{l_i\} \vdash B : K^n \\ \multicolumn{2}{c}{\Gamma_2, x :^{\omega} \{l_i\}, y :^{n} B \vdash \textbf{case}\, x\, \textbf{of}\, \{\overline{l_i \to N}\} : C} \\ \multicolumn{2}{c}{\downarrow\Gamma \vdash C : K'} \end{array}}{\Gamma \vdash \textbf{let}\, \langle x, y \rangle = M\, \textbf{in}\, N : C}$$

Results

- Embedding GV
- Soundness
- Progress
- Decidable type checking (subtyping, type equivalence)
- Type checker implemented (extended with recursive types)

Thank you!