Label-dependent Session Types

Peter Thiemann
Vasco T. Vasconcelos

University of Fribourg
University of Lisbon

ABCD meeting
London, December 2018

The good old math server

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

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
```

The I/O nature of channel operations

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

First-class labels

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

The pre-syntax of types

$$
\begin{array}{rl}
(x{:}\,A) \to B & \qquad \{l_1, \ldots, l_n\} \\
(x{:}\,A) \times B & \qquad \textbf{case } M \textbf{ of } \{l_i \to A_i\} \\
(x{:}\,A)\ !\ B & \qquad \textbf{Unit} \\
(x{:}\,A)\ ?\ B & \qquad M = N
\end{array}
$$

The label-dependent math server

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

The label-dependent math server

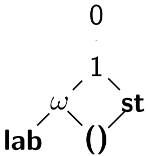
```
type LServer =
  (l: {Neg, Add}) ? case l of
    Neg → Int?Int!Unit
    Add → Int?Int?Int!Unit

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

Multiplicities

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

Multiplicities

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



Multiplicities

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

$$
\begin{tikzcd}
 & & 0 \arrow[d, dash, dotted] & \\
 & & 1 \arrow[dl, dash] \arrow[dr, dash] & \\
 & \omega \arrow[dl, dash] \arrow[dr, dash] & & \mathbf{st} \arrow[dl, dash] \\
\mathbf{lab} & & \mathbf{()} &
\end{tikzcd}
$$

demotion $\quad {\downarrow}0 = 0 \qquad {\downarrow}1 = 0 \qquad {\downarrow}\omega = \omega$

Output formation & elimination

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

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

Input formation & elimination

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

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

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 set of labels

Case formation & case introduction; label elimination

$$\frac{\Gamma_1 \vdash M : \{l_i\} \qquad \Gamma_2, \_ :^{\omega} M = l_i \vdash A_i : K \quad (\forall i)}{\Gamma_1 + \Gamma_2 \vdash \textbf{case}\, M\, \textbf{of}\, \{l_i \to A_i\} : K}$$

$$\frac{\Gamma_1 \vdash M : \{l_i\} \qquad \Gamma_2, \_ :^{\omega} M = l_i \vdash N_i : A \quad (\forall i)}{\Gamma_1 + \Gamma_2 \vdash \textbf{case}\, M\, \textbf{of}\, \{l_i \to N_i\} : A}$$

Term equality as a type

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A \qquad \Gamma \vdash A : \mathbf{lab}}{\Gamma \vdash M = N : \mathbf{un}}$$

No introduction or elimination rules
Instead, type $M = N$ is inhabited by the evidence that the values of M and N 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)
```

A sort of distributivity of **send**/**recv** over **case**

Can we type lServer against LServerR?

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

Can we type lServer against LServerR?

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

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

Can we type lServer against LServerR?

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

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

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

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 →

Can we type lServer against LServerR?

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

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

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

case l **of**

  Neg →

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

Can we type lServer against LServerR?

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

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

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

```
case l of
  Neg →
```

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

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

Can we type lServer against LServerR?

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

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

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

case l **of**

Neg →

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

let x, c = **recv** c

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

Can we type lServer against LServerR?

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

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

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

case l **of**

Neg →

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

let x, c = **recv** c

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

send c (−x) −− *we need c: Int !Unit*

Can we type lServer against LServerR?

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

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

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

case l **of**

Neg →

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

let x, c = **recv** c

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

send c (−x) −− *we need c: Int!Unit*

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

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 →

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

let x, c = **recv** c

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

send c (−x) *−− we need c: Int !Unit*

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

We need: **case** l **of** C ≡ **case** Neg **of** C ≡ **Int** ! **Unit**

Type Equivalence

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

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

Can we type lServerR against LServer?

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

Can we type lServerR against LServer?

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

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

Can we type lServerR against LServer?

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

let l, c = **recv** c

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

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 ...*

Can we type lServerR against LServer?

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

let l, c = **recv** c

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

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

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

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 ...*

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

case l **of**
  Neg →

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 ...*

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

case l **of**

Neg →

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

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 ...*

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

case l **of**

Neg →

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

send c (−x)

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 ...*

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

case l **of**

Neg →

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

send c (−x)

$$l :^\omega L, x :^\omega \mathbf{Int}, ;^\omega l = \mathrm{Neg}, c :^\omega \mathbf{Unit}$$

Type Equivalence

$$\frac{\Gamma \vdash \_ : M = N}{\Gamma \vdash \textbf{case}\, M\, \textbf{of}\, \{l_i \to A_i\} \equiv \textbf{case}\, N\, \textbf{of}\, \{l_i \to A_i\}}$$

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

$$\frac{}{\Gamma \vdash \textbf{case}\, M\, \textbf{of}\, \{l_i \to (x : A)?B_i\} \equiv (x : A)?\textbf{case}\, M\, \textbf{of}\, \{l_i \to B_i\}}$$

Similar rules for $(x : A)!B$, $(x : A) \to B$, and $(x : A) \times B$

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}) ×
  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

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

Typing sendEither

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

let tag, v = m **in**

Typing sendEither

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

let tag, v = m **in**

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

Typing sendEither

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

let tag, v = m **in**

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

let c = **send** c tag

Typing sendEither

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

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

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

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

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

```
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
```

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

```
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

$$\frac{\downarrow \Gamma \vdash M : \{l_i\} \qquad \Gamma, \_ :^{\omega} M = l_i \vdash N : A \quad (\forall i)}{\Gamma \vdash N : A}$$

Results

- Embedding GV
- Soundness
- Progress

Thank you!