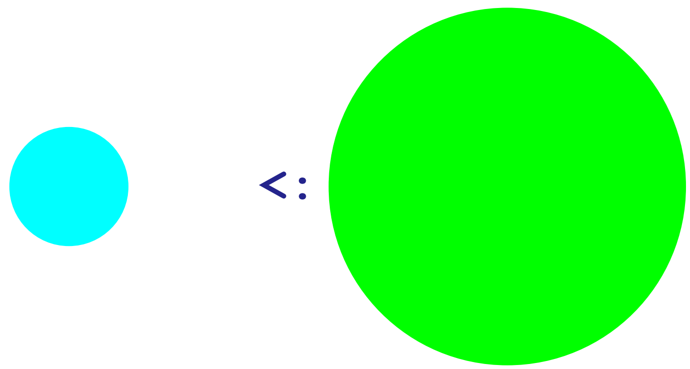
Part 1

Subtyping



PosInt <: Int

Typed Records

```
<Exp>  ::=  <Int>
         |  <Exp> + <Exp>
         |  <Exp> * <Exp>
         |  <Symbol>
         |  fun (<Symbol> :: <Type>): <Exp>
         |  <Exp>(<Exp>)
         |  { <Symbol>: <Exp>, ... }                NEW
         |  <Exp>.<Symbol>                          NEW
         |  <Exp> with (<Symbol> = <Exp>)           NEW

<Type> ::=  Int
         |  Boolean
         |  <Type> -> <Type>
         |  { <Symbol> :: <Type>, ... }             NEW
```

Records

```
{ x: 1 + 2,
  y: 3 * 4 }
```

Has type
```
{ x :: Int,
  y :: Int }
```

Records

```
{ p: { x: 1 + 2,
       y: 3 * 4 } }
```

Has type
```
{ p :: { x :: Int,
         y :: Int } }
```

Records

```
{ x: 1 + 2,
  y: 3 * 4 }.y
```

The subexpression type

```
{ x :: Int,
  y :: Int }
```

means that `.` will succeed for `y`

Records

```
{ p: { x: 1 + 2,
       y: 3 * 4 } }.p.x
```

The subexpression type

```
{ p :: { x :: Int,
         y :: Int } }
```

means that `.` will succeed for `p` then `x`

Records

```
{ x: 1 + 2,
  y: 3 * 4 } with (y = 0)
```

Same type and value as

```
{ x: 3,
  y: 0 }
```

Records

```
let r :: { x :: Int,
           y :: Int } = { x: 1 + 2,
                          y: 3 * 4 }:
  r.x + r.y
```

Records

```
fun (r :: { x :: Int }):
  r.x
```

Has type `{ x :: Int } -> Int`

Records

```
fun (r :: { x :: Int }):
  r with (x = 1)
```

Has type `{ x :: Int } -> { x :: Int }`

Records

```
let f :: { x :: Int } -> { x :: Int } = (fun (r :: { x :: Int }):
                                             r with (x = 1)):
  f({ x: 1 + 1 })
```

Typechecking

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma \vdash e_n : \tau_n \qquad \tau = \texttt{\{}\ x_1\ \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{::}\ \tau_n\ \texttt{\}}}{\Gamma \vdash \texttt{\{}\ x_1\ \texttt{:}\ e_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{:}\ e_n\ \texttt{\}} : \tau}$$

$$\frac{\Gamma \vdash e : \texttt{\{}\ x_1\ \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{::}\ \tau_n\ \texttt{\}} \qquad x_i \in \{x_1, \ldots, x_n\}}{\Gamma \vdash e\texttt{.}x_i : \tau_i}$$

$$\frac{\Gamma \vdash e_1 : \texttt{\{}\ x_1\ \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{::}\ \tau_n\ \texttt{\}} \qquad \Gamma \vdash e_2 : \tau_i \qquad x_i \in \{x_1, \ldots, x_n\}}{\Gamma \vdash e_1\ \texttt{with}\ \texttt{(}x_i\ \texttt{=}\ e_2\texttt{)} : \texttt{\{}\ x_1\ \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{::}\ \tau_n\ \texttt{\}}}$$

Part 2

Records and Fields

```
(fun (r :: { x :: Int }): r.x)({ x: 1 })
```

Records and Fields

```
{ x :: Int } -> Int
```

```
(fun (r :: { x :: Int }): r.x)({ x: 1 })
```

Records and Fields

```
{ x :: Int } -> Int
(fun (r :: { x :: Int }): r.x)({ x: 1 })
{ x :: Int }
```

Records and Fields

`{ x :: Int } -> Int`

```
(fun (r :: { x :: Int }): r.x)({ x: 1 })
```

`{ x :: Int }`

Has type `Int`

$$\frac{\Gamma \vdash \mathbf{e}_1 : \tau_2 \texttt{ -> } \tau_3 \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2}{\Gamma \vdash \mathbf{e}_1 \texttt{ (} \mathbf{e}_2 \texttt{) } : \tau_3}$$

Records and Fields

```
(fun (r :: { x :: Int, y :: Int }): r.x)({ y: 1,
                                           x: 2 })
```

Records and Fields

```
{ x :: Int, y :: Int } -> Int
```

```
(fun (r :: { x :: Int, y :: Int }): r.x)({ y: 1,
                                           x: 2 })
```

Records and Fields

```
{ x :: Int, y :: Int } -> Int
```

```
(fun (r :: { x :: Int, y :: Int }): r.x)({ y: 1,
                                           x: 2 })
```

```
{ y :: Int, x :: Int }
```

Records and Fields

```
{ x :: Int, y :: Int } -> Int
```

```
(fun (r :: { x :: Int, y :: Int }): r.x)({ y: 1,
                                           x: 2 })
```

```
{ y :: Int, x :: Int }
```

no type — field order doesn't match

$$\frac{\Gamma \vdash \mathbf{e}_1 : \tau_2 \texttt{ -> } \tau_3 \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2}{\Gamma \vdash \mathbf{e}_1 \texttt{(} \mathbf{e}_2 \texttt{)} : \tau_3}$$

Records and Fields

```
(fun (r :: { x :: Int }): r.x)({ x: 1,
                                 y: 2 })
```

Records and Fields

```
{ x :: Int } -> Int
```

```
(fun (r :: { x :: Int }): r.x)({ x: 1,
                                 y: 2 })
```

Records and Fields

```
{ x :: Int } -> Int
```

```
(fun (r :: { x :: Int }): r.x)({ x: 1,
                                  y: 2 })
```

```
{ x :: Int, y :: Int }
```

Records and Fields

```
{ x :: Int } -> Int
```

```
(fun (r :: { x :: Int }): r.x)({ x: 1,
                                 y: 2 })
```

```
{ x :: Int, y :: Int }
```

no type — extra fields in argument

$$\frac{\Gamma \vdash \mathbf{e}_1 : \tau_2 \;\texttt{->}\; \tau_3 \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2}{\Gamma \vdash \mathbf{e}_1 \; (\; \mathbf{e}_2 \;) : \tau_3}$$

Subtypes

If τ is a ***subtype*** of τ',

$$\tau \leq \tau'$$

then an expression of type τ can be used in place of an expression of type τ'

`{ x :: Int, y :: Int }` ≤ `{ x :: Int }`

Subtypes

If τ is a ***subtype*** of τ',

$$\tau \leq \tau'$$

then an expression of type τ can be used in place of an expression of type τ'

`{ y :: Int, x :: Int }` ≤ `{ x :: Int, y :: Int }`

Subtypes

If τ is a ***subtype*** of τ',

$$\tau \leq \tau'$$

then an expression of type τ can be used in place of an expression of type τ'

`{ x :: Int }` ≤ `{ x :: Int }`

Subtypes

If τ is a ***subtype*** of τ',

$$\tau \leq \tau'$$

then an expression of type τ can be used in place of an expression of type τ'

`{ x :: Int }` $\not\leq$ `{ x :: Int, y :: Int }`

Subtypes

If τ is a ***subtype*** of τ',

$$\tau \leq \tau'$$

then an expression of type τ can be used in place of an expression of type τ'

`{ x :: Int, y :: Int }` ≤ `{ x :: Int }`

Intution: $\tau \leq \tau'$ means that fewer values fit τ than τ'

Subtype Rules

$$\frac{\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \supseteq \{\mathbf{x}'_1, \ldots, \mathbf{x}'_m\} \qquad \mathbf{x}_i = \mathbf{x}'_j \Rightarrow \tau_i = \tau'_j}{\texttt{\{}\ \mathbf{x}_1\ \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}_n\ \texttt{::}\ \tau_n\ \texttt{\}} \le \texttt{\{}\ \mathbf{x}'_1\ \texttt{::}\ \tau'_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}'_m\ \texttt{::}\ \tau'_m\ \texttt{\}}}$$

$$\texttt{Int} \le \texttt{Int} \qquad \texttt{Boolean} \le \texttt{Boolean}$$

$$\tau_1\ \texttt{->}\ \tau_2 \le \tau_1\ \texttt{->}\ \tau_2$$

$$\frac{\Gamma \vdash \mathbf{e}_1 : \tau_1\ \texttt{->}\ \tau_3 \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2 \qquad \tau_2 \le \tau_1}{\Gamma \vdash \mathbf{e}_1\ \texttt{(}\ \mathbf{e}_2\ \texttt{)} : \tau_3}$$

Records and Fields

```
(fun (r :: { x :: Int }): r.x)({ x: 1,
                                  y: 2 })
```

Records and Fields

```
{ x :: Int } -> Int
```

```
(fun (r :: { x :: Int }): r.x)({ x: 1,
                                y: 2 })
```

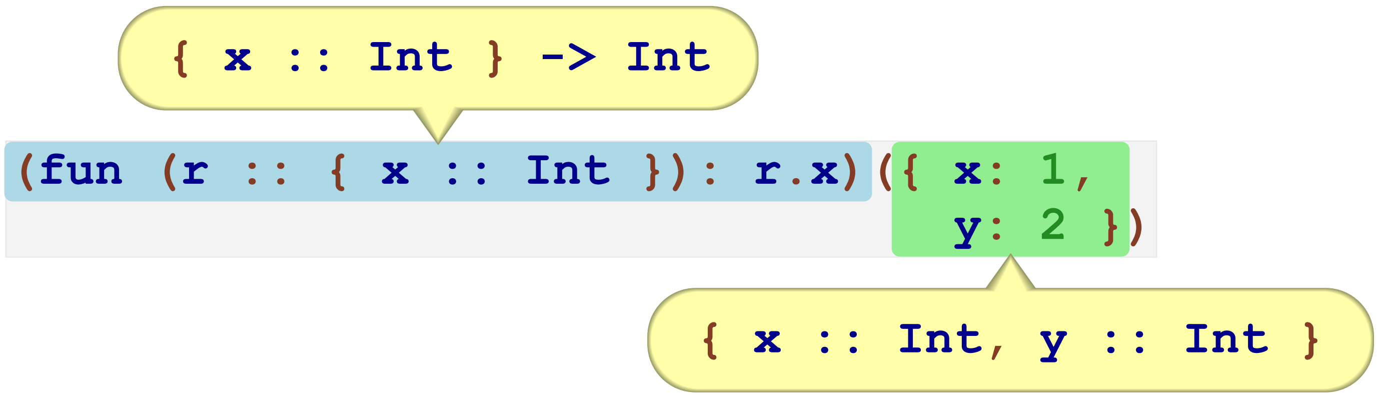
Records and Fields

```
{ x :: Int } -> Int
```

```
(fun (r :: { x :: Int }): r.x)({ x: 1,
                                  y: 2 })
```

```
{ x :: Int, y :: Int }
```

Records and Fields



`{ x :: Int, y :: Int }` ≤ `{ x :: Int }`

Records and Fields

```
(fun (r :: { y :: Int, x :: Int }): r.x)({ x: 1,
                                            y: 2 })
```

Records and Fields

```
{ y :: Int, x :: Int } -> Int
```

```
(fun (r :: { y :: Int, x :: Int }): r.x)({ x: 1,
                                           y: 2 })
```

Records and Fields

```
{ y :: Int, x :: Int } -> Int
```

```
(fun (r :: { y :: Int, x :: Int }): r.x)({ x: 1,
                                            y: 2 })
```

```
{ x :: Int, y :: Int }
```

Records and Fields

```
{ y :: Int, x :: Int } -> Int
```

```
(fun (r :: { y :: Int, x :: Int }): r.x)({ x: 1,
                                            y: 2 })
```

```
{ x :: Int, y :: Int }
```

`{ x :: Int, y :: Int }` ≤ `{ y :: Int, x :: Int }`

Records and Fields

```
(fun (r :: { x :: Int, y :: Int }): r.x)({ y: 5 })
```

Records and Fields

```
{ x :: Int, y :: Int } -> Int
```

```
(fun (r :: { x :: Int, y :: Int }): r.x)({ y: 5 })
```

Records and Fields

```
(fun (r :: { x :: Int, y :: Int }): r.x)({ y: 5 })
```

```
{ x :: Int, y :: Int } -> Int
```

```
{ y :: Int }
```

Records and Fields

{ x :: Int, y :: Int } -> Int

```
(fun (r :: { x :: Int, y :: Int }): r.x)({ y: 5 })
```

{ y :: Int }

$\{ y :: Int \} \not\leq \{ x :: Int, y :: Int \}$

Part 3

Subtypes in Fields

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                   r.p.x):
  f({ p: { x: 5,
           y: 6 } })
```

Subtypes in Fields

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                   r.p.x):
  f({ p: { x: 5,
           y: 6 } })
```

```
{ p :: { x :: Int } }
```

Subtypes in Fields

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                  r.p.x):
  f({ p: { x: 5,
           y: 6 } })
```

```
{ p :: { x :: Int,
         y :: Int } }
```

vs.

```
{ p :: { x :: Int } }
```

Subtypes in Fields

```
{ p :: { x :: Int,
         y :: Int } }
```

vs.

```
{ p :: { x :: Int } }
```

$$\frac{\{\mathbf{x}_1,\ \ldots,\ \mathbf{x}_n\} \supseteq \{\mathbf{x}'_1,\ \ldots,\ \mathbf{x}'_m\} \qquad \mathbf{x}_i = \mathbf{x}'_j \Rightarrow \tau_i = \tau'_j}{\{\ \mathbf{x}_1 :: \tau_1,\ \ldots,\ \mathbf{x}_n :: \tau_n\ \} \leq \{\ \mathbf{x}'_1 :: \tau'_1,\ \ldots,\ \mathbf{x}'_m :: \tau'_m\ \}}$$

Subtypes in Fields

```
{ p :: { x :: Int,
         y :: Int } }
```

vs.

```
{ p :: { x :: Int } }
```

$$\frac{\{x_1, \ldots, x_n\} \supseteq \{x'_1, \ldots, x'_m\} \qquad x_i = x'_j \Rightarrow \tau_i \le \tau'_j}{\texttt{\{}\ x_1\ \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{::}\ \tau_n\ \texttt{\}} \le \texttt{\{}\ x'_1\ \texttt{::}\ \tau'_1\texttt{,}\ \ldots\texttt{,}\ x'_m\ \texttt{::}\ \tau'_m\ \texttt{\}}}$$

Field Update and Subtypes

```
fun (r :: { p :: { x :: Int } }):
  r with (p = { x: 5,
                y: 6 })
```

Field Update and Subtypes

Original rule:

$$\frac{\Gamma \vdash \mathbf{e}_1 : \texttt{\{}\ \mathbf{x}_1 \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}_n \texttt{::}\ \tau_n\ \texttt{\}} \quad \Gamma \vdash \mathbf{e}_2 : \tau_i \quad \mathbf{x}_i \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}}{\Gamma \vdash \mathbf{e}_1\ \texttt{with}\ \texttt{(}\mathbf{x}_i\ \texttt{=}\ \mathbf{e}_2\texttt{)} : \texttt{\{}\ \mathbf{x}_1 \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}_n \texttt{::}\ \tau_n\ \texttt{\}}}$$

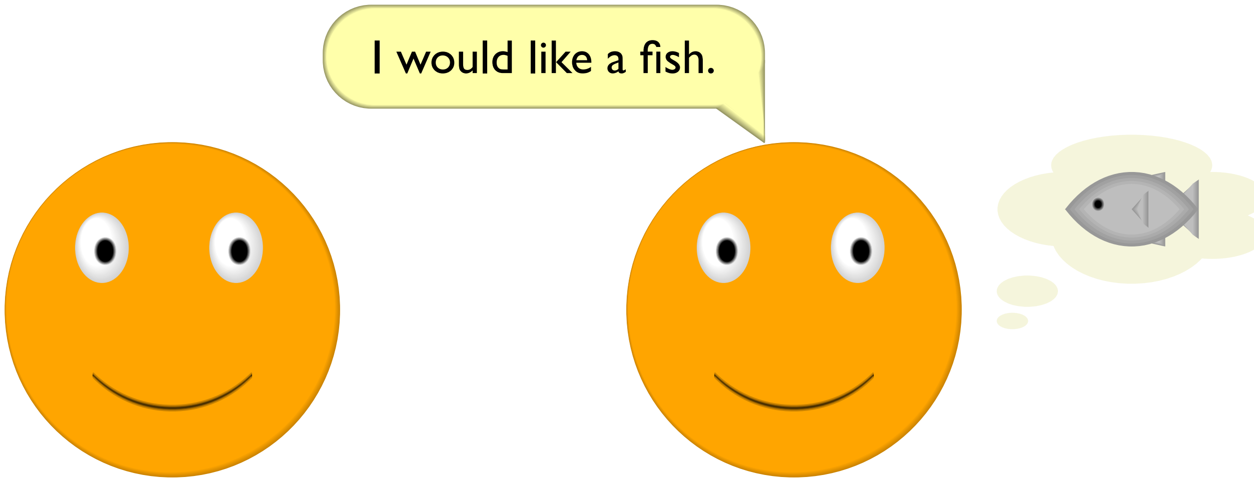
Revised rule:

$$\frac{\Gamma \vdash \mathbf{e}_1 : \texttt{\{}\ \mathbf{x}_1 \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}_n \texttt{::}\ \tau_n\ \texttt{\}} \quad \Gamma \vdash \mathbf{e}_2 : \tau' \quad \mathbf{x}_i \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \quad \tau' \leq \tau_i}{\Gamma \vdash \mathbf{e}_1\ \texttt{with}\ \texttt{(}\mathbf{x}_i\ \texttt{=}\ \mathbf{e}_2\texttt{)} : \texttt{\{}\ \mathbf{x}_1 \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}_n \texttt{::}\ \tau_n\ \texttt{\}}}$$

Part 4

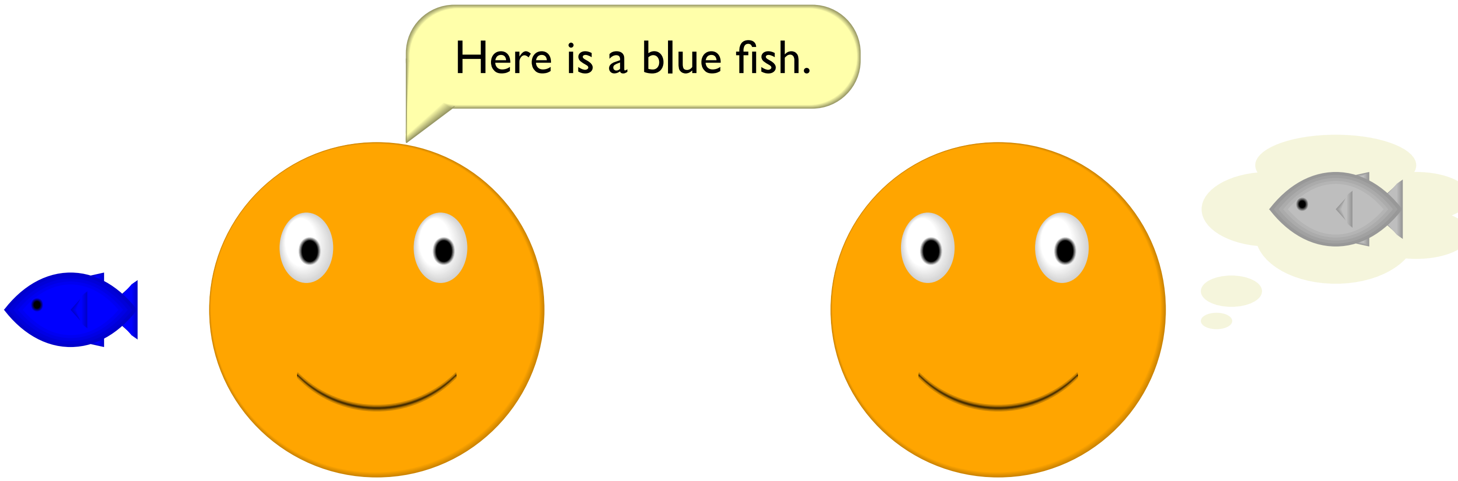
Subtypes and Functions

Subtypes and Functions




```
{ sz :: Int }
```

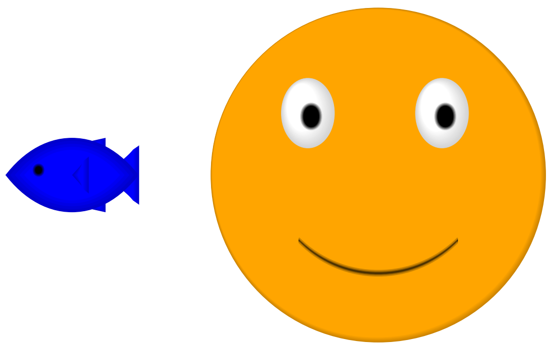
Subtypes and Functions

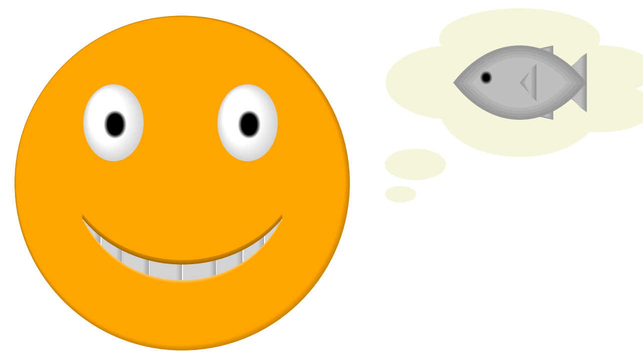


`{ sz :: Int }`

`{ sz :: Int, col :: Int }`

Subtypes and Functions





`{ sz :: Int }`

`{ sz :: Int, col :: Int }`

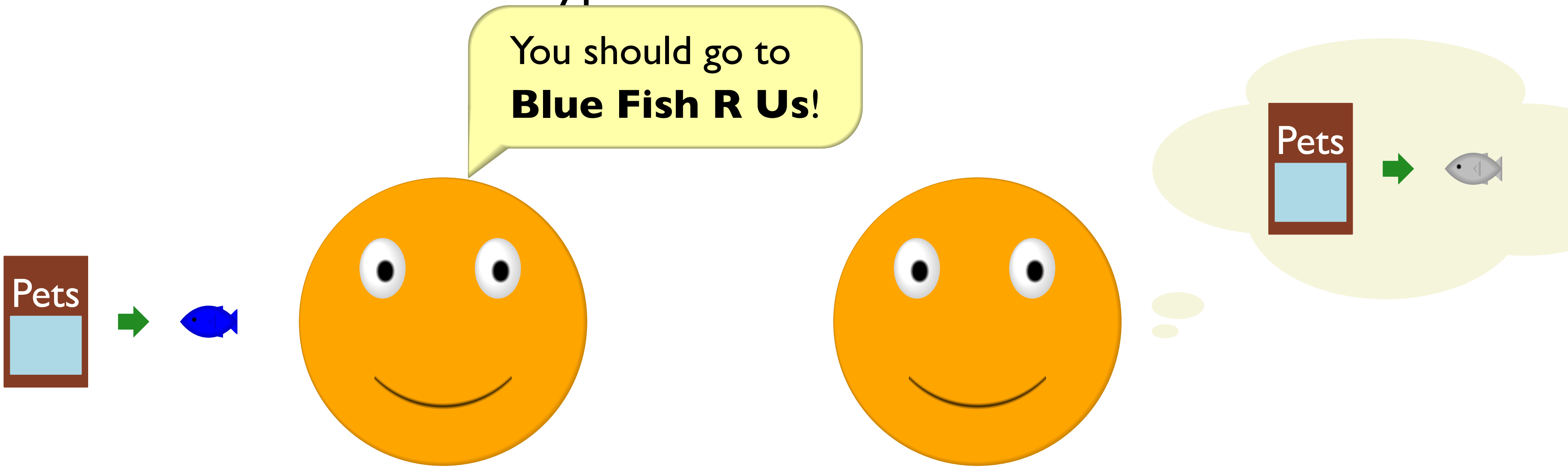
`{ sz :: Int, col :: Int }` ≤ `{ sz :: Int }`

Subtypes and Functions

Subtypes and Functions

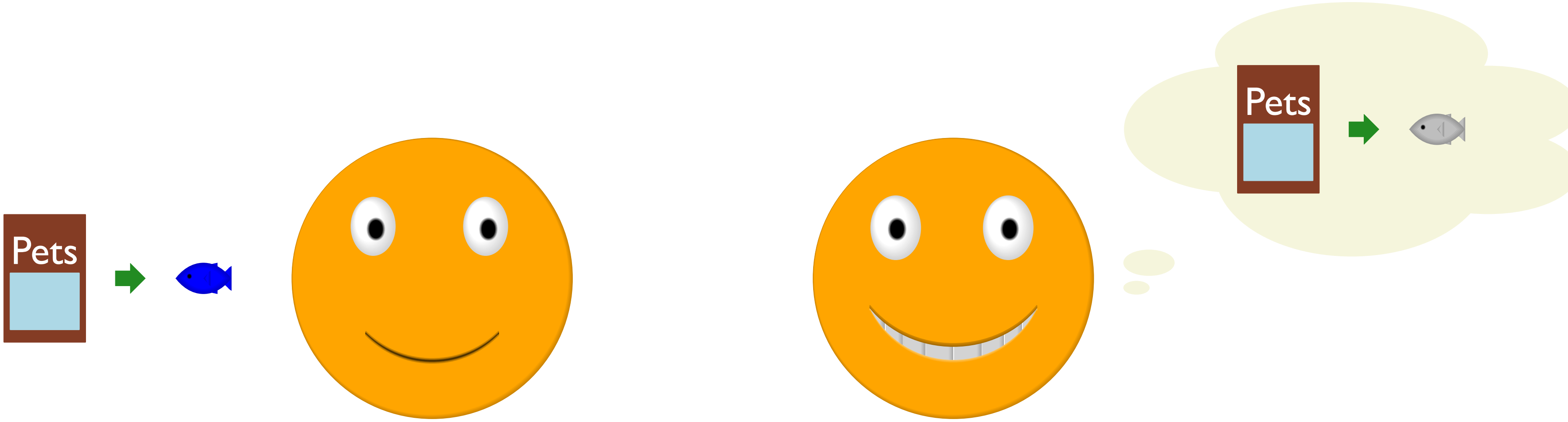

```
Int -> { sz :: Int }
```

Subtypes and Functions




```
Int -> { sz :: Int }
Int -> { sz :: Int, col :: Int }
```

Subtypes and Functions



`Int -> { sz :: Int }`

`Int -> { sz :: Int, col :: Int }`

`Int -> { sz :: Int, col :: Int }` ≤ `Int -> { sz :: Int }`

Subtypes from Functions

```
let f :: .... = (fun (g :: Int -> { x :: Int }):
                  g(10).x):
  f(fun (v :: Int):
      { x: v,
        y: v })
```

Subtypes from Functions

```
let f :: .... = (fun (g :: Int -> { x :: Int }):
                         g(10).x):
  f(fun (v :: Int):
        { x: v,
          y: v })
```

```
Int -> { x :: Int }
```

Subtypes from Functions

```
let f :: .... = (fun (g :: Int -> { x :: Int }) :
                         g(10).x) :
  f(fun (v :: Int):
      { x: v,
        y: v })
```

```
Int -> { x :: Int, y :: Int }
```

vs.

```
Int -> { x :: Int }
```

Subtypes from Functions

```
Int -> { x :: Int, y :: Int }
```

vs.

```
Int -> { x :: Int }
```

τ_1 `->` τ_2 $\leq$ τ_1 `->` τ_2

Subtypes from Functions

```
Int -> { x :: Int, y :: Int }
```

vs.

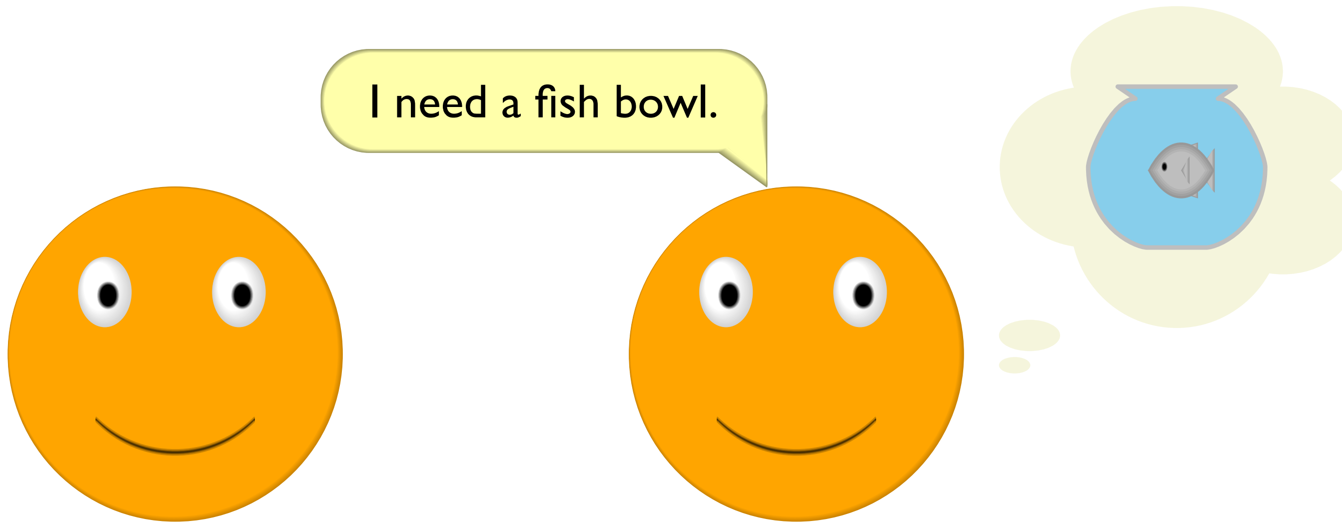
```
Int -> { x :: Int }
```

$$\frac{\tau_2 \leq \tau_2'}{\tau_1 \texttt{ -> } \tau_2 \leq \tau_1 \texttt{ -> } \tau_2'}$$

Part 5

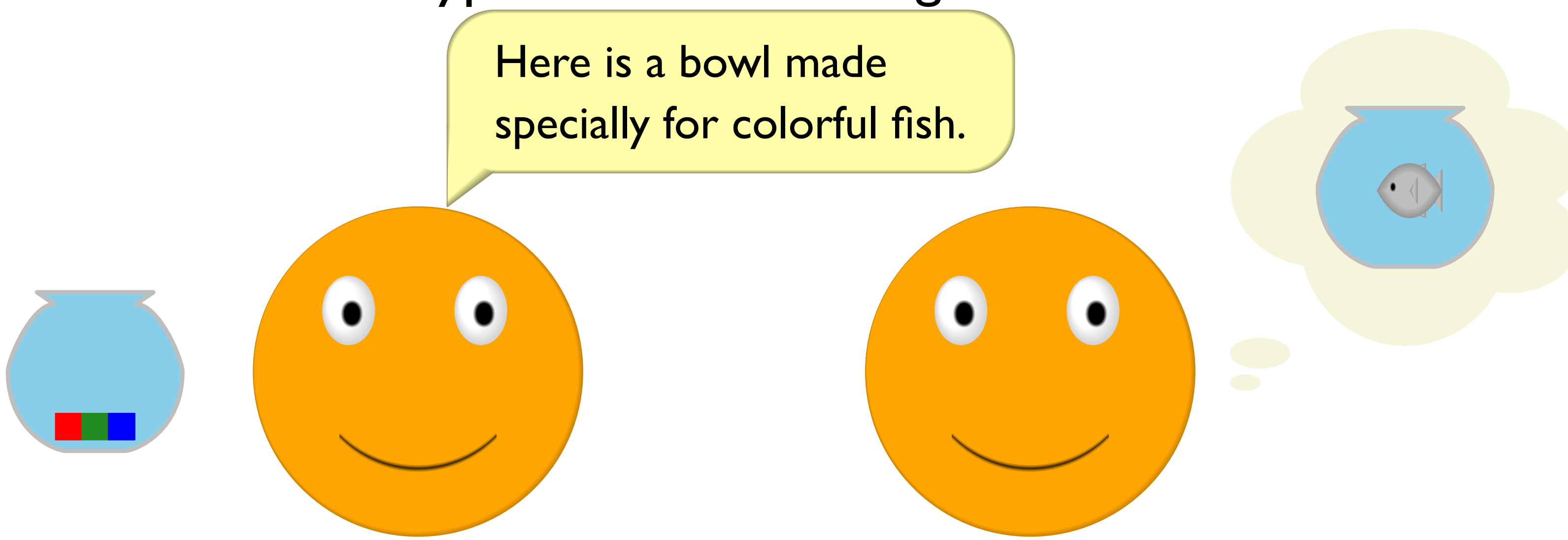
Subtype and Function Arguments

Subtype and Function Arguments




```
{ sz :: Int } -> Int
```

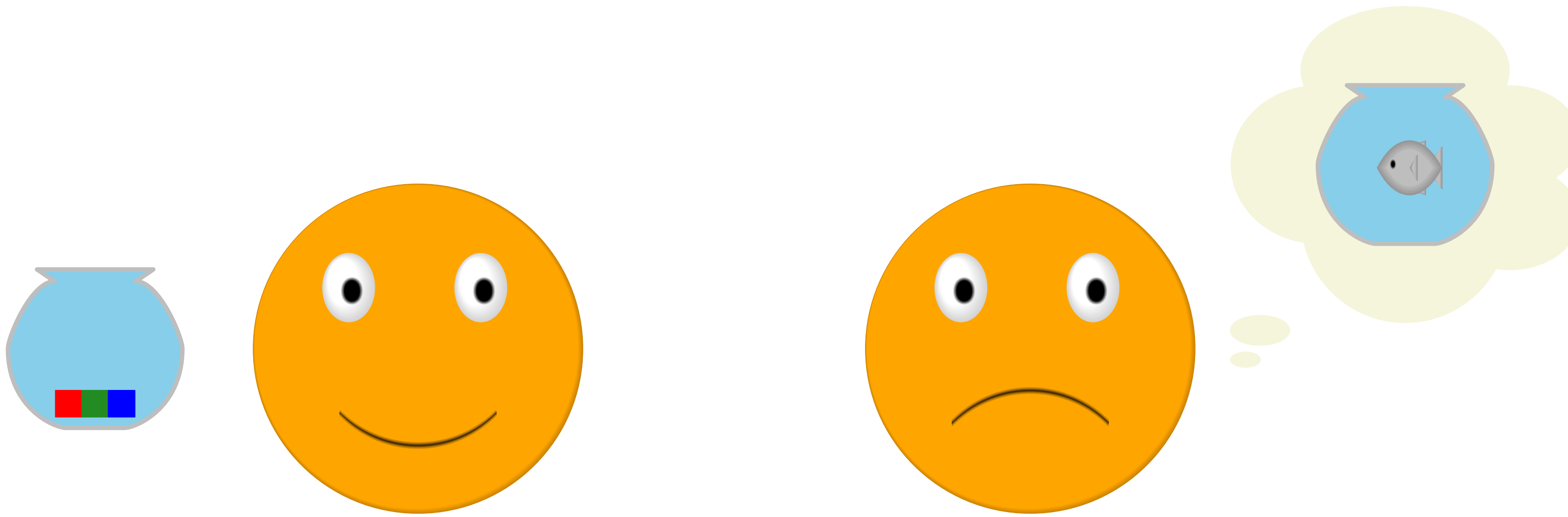
Subtype and Function Arguments




```
{ sz :: Int } -> Int
```

```
{ sz :: Int, col :: Int } -> Int
```

Subtype and Function Arguments



`{ sz :: Int } -> Int`

`{ sz :: Int, col :: Int } -> Int`

`{ sz :: Int, col :: Int } -> Int` $\not\leq$ `{ sz :: Int } -> Int`

Subtypes and Function Arguments

```
let f :: ? = (fun (g :: { x :: Int } -> Int):
                  g({ x: 1 })):
  f(fun (r :: { x :: Int, y :: Int }):
      r.y)
```

Subtypes and Function Arguments

```
let f :: ? = (fun (g :: { x :: Int } -> Int):
                g({ x: 1 })):
  f(fun (r :: { x :: Int, y :: Int }):
      r.y)
```

```
{ x :: Int } -> Int
```

Subtypes and Function Arguments

```
let f :: ? = (fun (g :: { x :: Int } -> Int):
                          g({ x: 1 })):
  f(fun (r :: { x :: Int, y :: Int }):
      r.y)
```

```
{ x :: Int, y :: Int } -> Int
```

vs.

```
{ x :: Int } -> Int
```

Subtypes and Function Arguments

```
{ x :: Int, y :: Int } -> Int
```

vs.

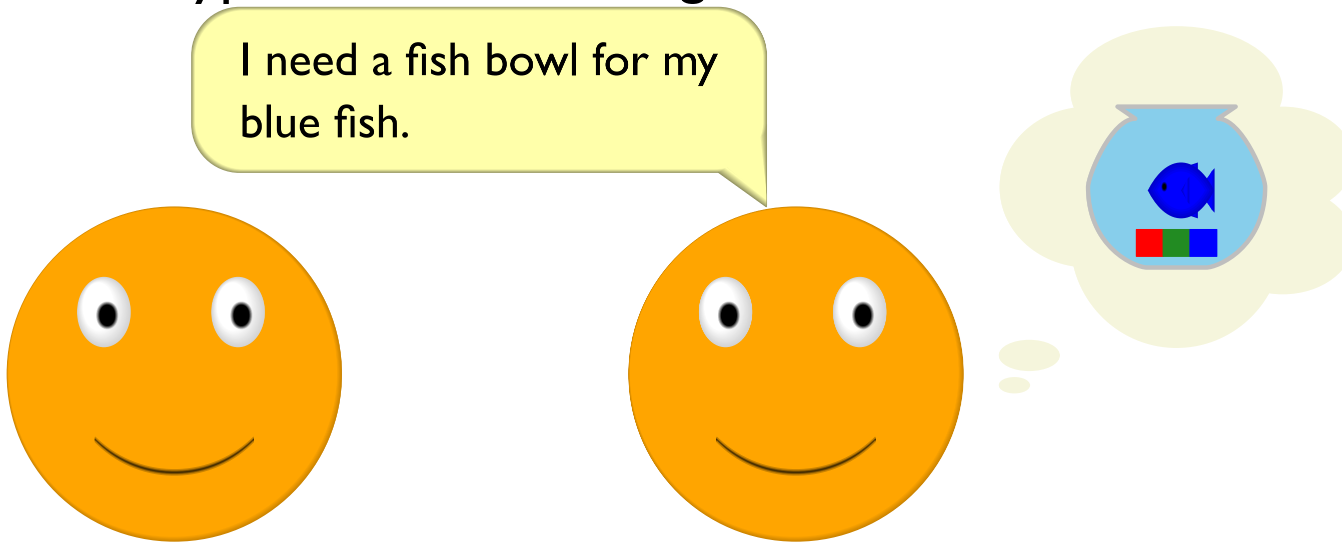
```
{ x :: Int } -> Int
```

$$\frac{\tau_2 \leq \tau_2'}{\tau_1 \texttt{ -> } \tau_2 \leq \tau_1 \texttt{ -> } \tau_2'}$$

Correctly rejected!

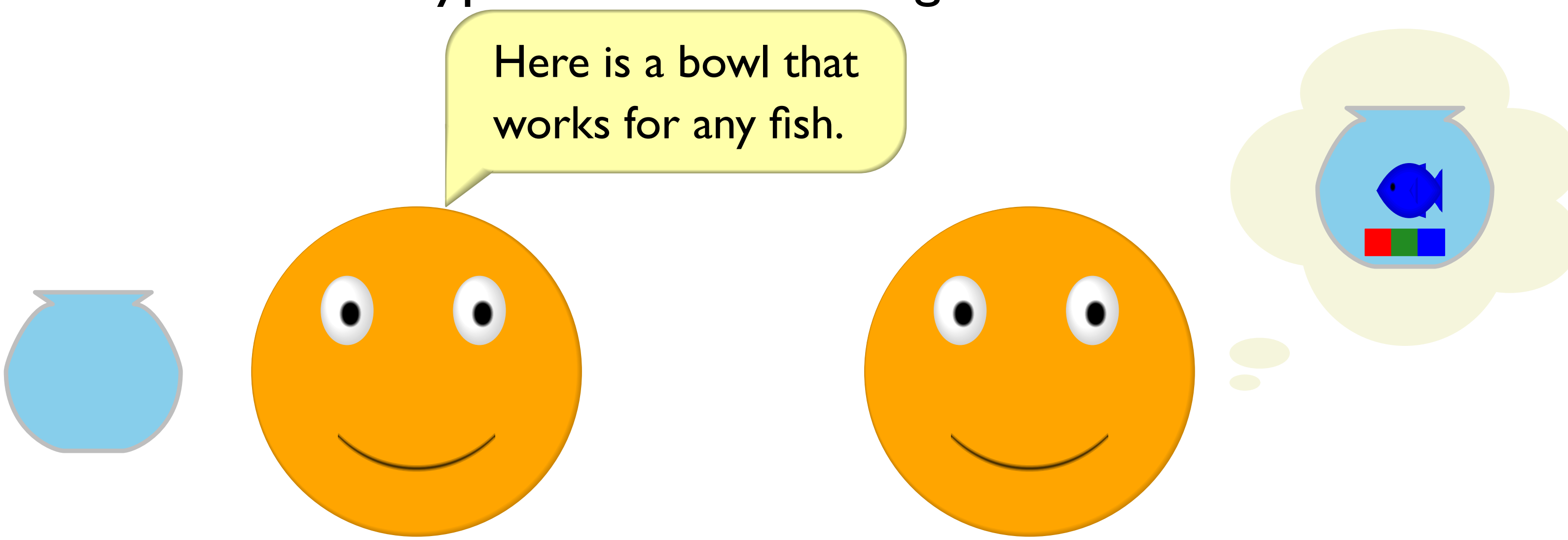
Subtype and Function Arguments

Subtype and Function Arguments




```
{ sz :: Int, col :: Int } -> Int
```

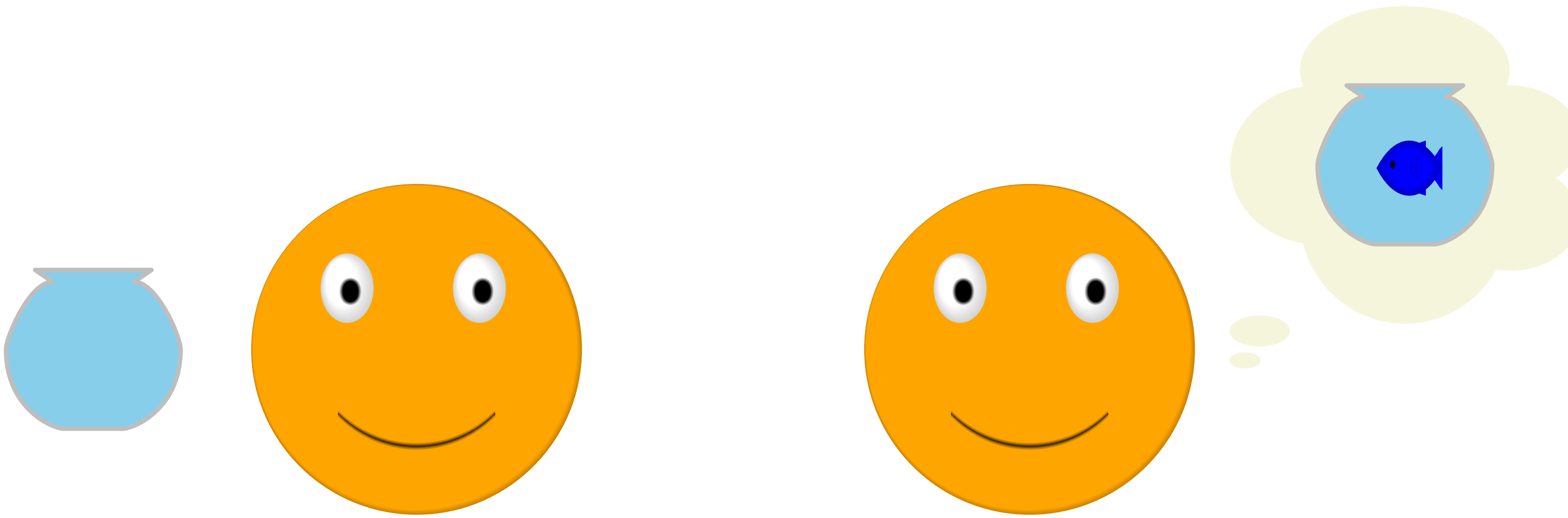
Subtype and Function Arguments




```
{ sz :: Int, col :: Int } -> Int
```

```
{ sz :: Int } -> Int
```

Subtype and Function Arguments



`{ sz :: Int, col :: Int } -> Int`

`{ sz :: Int } -> Int`

`{ sz :: Int } -> Int` ≤ `{ sz :: Int, col :: Int } -> Int`

Subtypes to Functions

```
let f :: .... = (fun (g :: { x :: Int, y :: Int } -> Int):
                         g({ x: 1,
                             y: 2 })):
  f(fun (r :: { x :: Int }):
      r.x)
```

Subtypes to Functions

```
let f :: .... = (fun (g :: { x :: Int, y :: Int } -> Int):
                 g({ x: 1,
                     y: 2 })):
  f(fun (r :: { x :: Int }):
      r.x)
```

```
{ x :: Int, y :: Int } -> Int
```

Subtypes to Functions

```
let f :: .... = (fun (g :: { x :: Int, y :: Int } -> Int):
                   g({ x: 1,
                       y: 2 })):
  f(fun (r :: { x :: Int }):
      r.x)
```

```
{ x :: Int } -> Int
```

vs.

```
{ x :: Int, y :: Int } -> Int
```

Subtypes to Functions

```
{ x :: Int } -> Int
```

vs.

```
{ x :: Int, y :: Int } -> Int
```

$$\frac{\tau_2 \leq \tau_2'}{\tau_1 \texttt{ -> } \tau_2 \leq \tau_1 \texttt{ -> } \tau_2'}$$

Subtypes to Functions

```
{ x :: Int } -> Int
```

vs.

```
{ x :: Int, y :: Int } -> Int
```

$$\frac{\tau_1' \leq \tau_1 \qquad \tau_2 \leq \tau_2'}{\tau_1 \texttt{ -> } \tau_2 \leq \tau_1' \texttt{ -> } \tau_2'}$$

Part 6

Covariance and Contravariance

$$\frac{\tau_1' \leq \tau_1 \qquad \tau_2 \leq \tau_2'}{\tau_1 \texttt{ -> } \tau_2 \leq \tau_1' \texttt{ -> } \tau_2'}$$

Function-result types are ***covariant*** with function types

Function-argument types are ***contravariant*** with function types

Covariance and Contravariance

$$\frac{\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \supseteq \{\mathbf{x}'_1, \ldots, \mathbf{x}'_m\} \qquad \mathbf{x}_i = \mathbf{x}'_j \Rightarrow \tau_i \leq \tau'_j}{\texttt{\{}\ \mathbf{x}_1\ \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}_n\ \texttt{::}\ \tau_n\ \texttt{\}} \leq \texttt{\{}\ \mathbf{x}'_1\ \texttt{::}\ \tau'_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}'_m\ \texttt{::}\ \tau'_m\ \texttt{\}}}$$

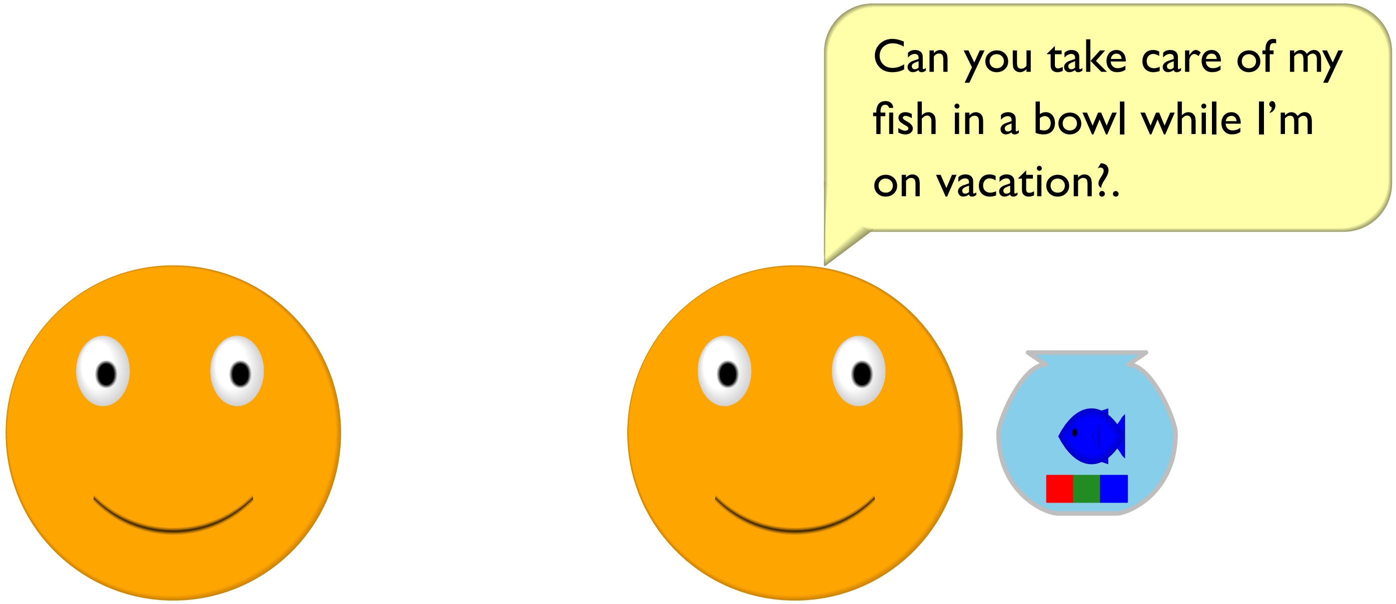
Field types are ***covariant*** with record types

... as long as `set` is a functional update

Part 7

Subtypes and Functions

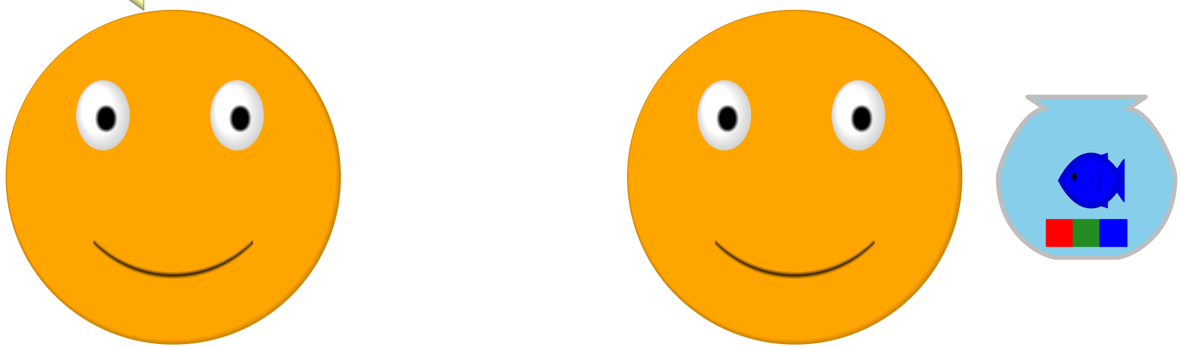
Subtypes and Functions




```
{ fish :: { sz :: Int,
            col :: Int } }
```

Subtypes and Functions

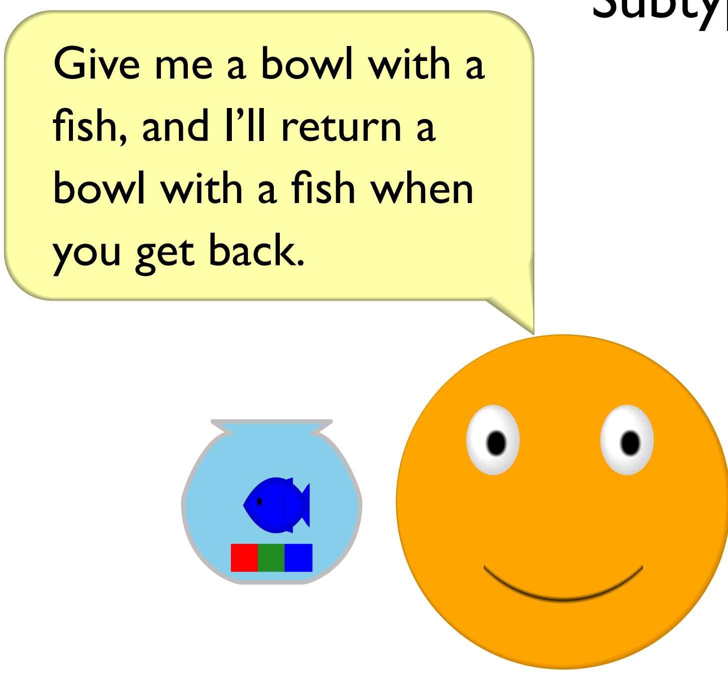
Give me a bowl with a fish, and I'll return a bowl with a fish when you get back.



```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> { fish :: { sz :: Int } }
```

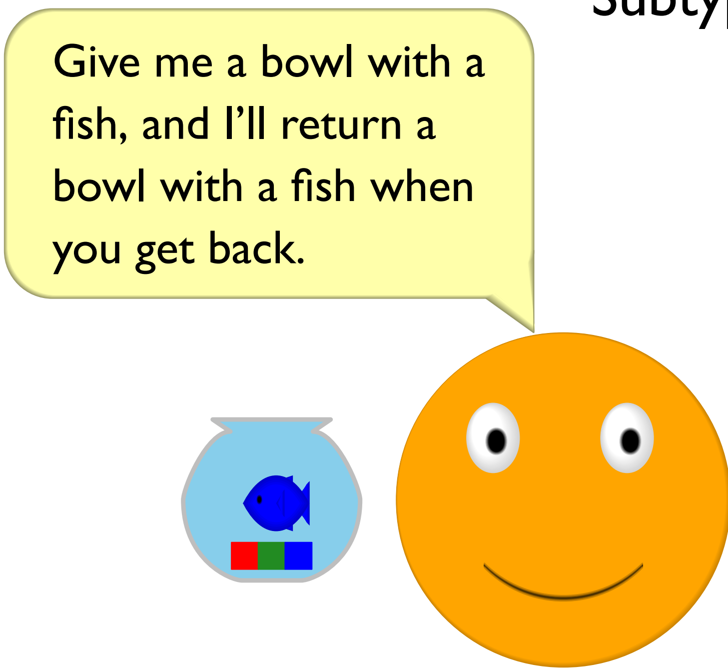
Subtypes and Functions



```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> { fish :: { sz :: Int } }
```

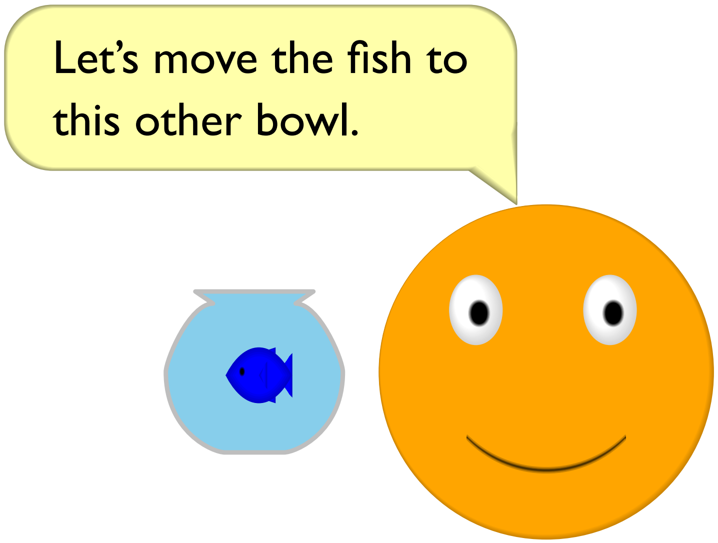
Subtypes and Functions




```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> { fish :: { sz :: Int } }
```

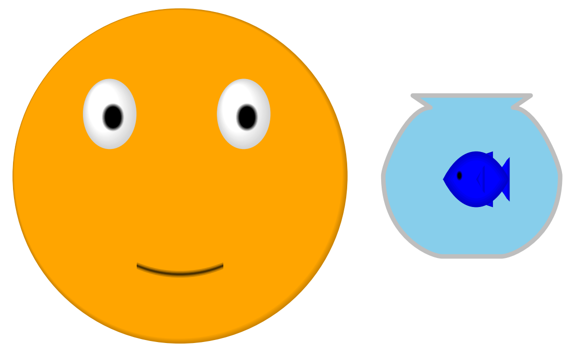
Subtypes and Functions




```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> { fish :: { sz :: Int } }
```

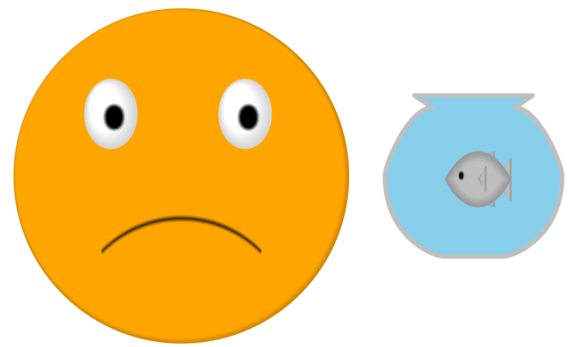
Subtypes and Functions



```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> { fish :: { sz :: Int } }
```

Subtypes and Functions

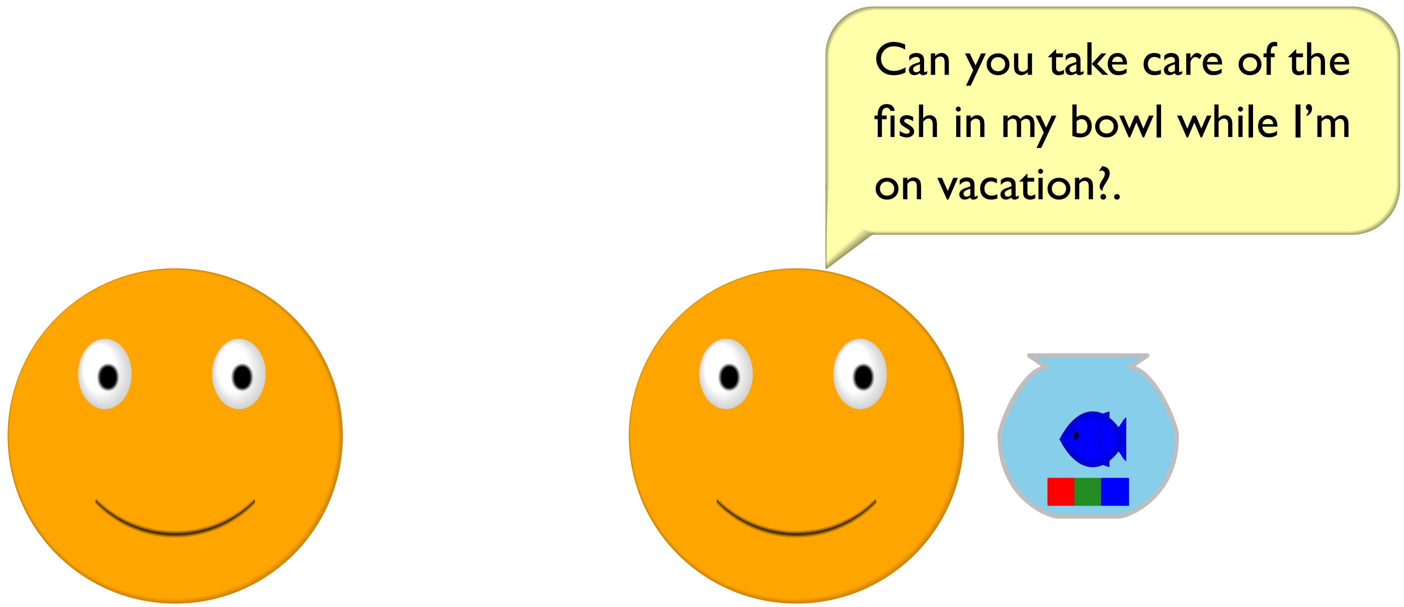


```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> { fish :: { sz :: Int } }
```

Subtypes and Functions

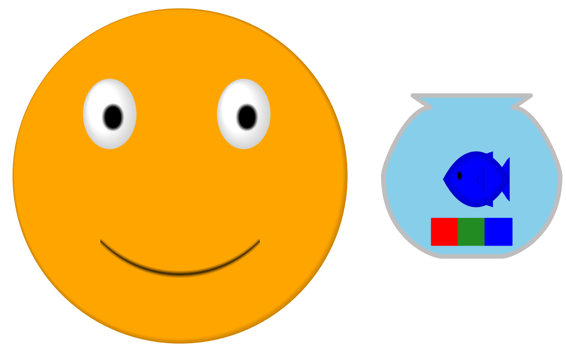
Subtypes and Functions




```
{ fish :: { sz :: Int,
            col :: Int } }
```

Subtypes and Functions

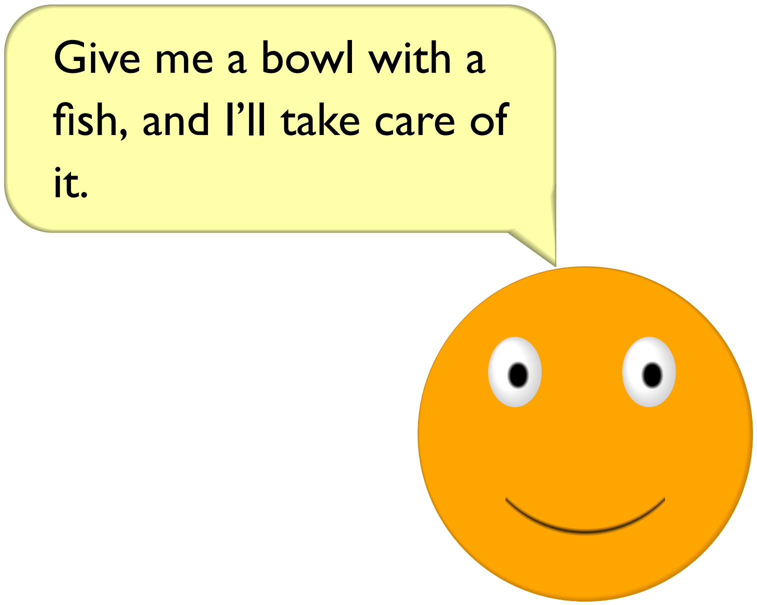
Give me a bowl with a fish, and I'll take care of it.

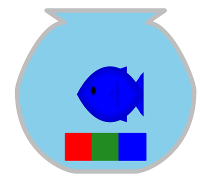


```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> bool
```

Subtypes and Functions

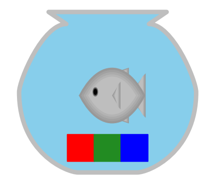




```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> bool
```

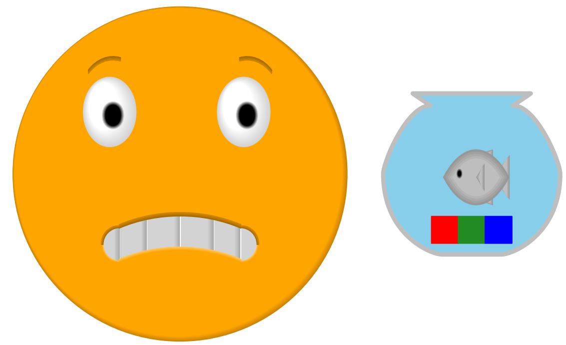
Subtypes and Functions



```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> bool
```

Subtypes and Functions



```
{ fish :: { sz :: Int,
            col :: Int } }
```

```
{ fish :: { sz :: Int } }
  -> bool
```

Subtypes and Update

```
let f :: .... = (fun (r :: { x :: Int }):
                    r with (x = 5)):
  f({ x: 1, y: 2 })
```

Subtypes and Update

```
let f :: .... = (fun (r :: { x :: Int }) :
                        r with (x = 5)) :
  f({ x: 1, y: 2 })
```

```
{ x :: Int }
```

Subtypes and Update

```
let f :: .... = (fun (r :: { x :: Int }):
                       r with (x = 5)):
  f({ x: 1, y: 2 })
```

```
{ x :: Int, y :: Int }
```

vs.

```
{ x :: Int }
```

Subtypes and Update

```
{ x :: Int, y :: Int }
```

vs.

```
{ x :: Int }
```

$$\frac{\{x_1, \ldots, x_n\} \supseteq \{x'_1, \ldots, x'_m\} \qquad x_i = x'_j \Rightarrow \tau_i \leq \tau'_j}{\{\ x_1 :: \tau_1, \ldots, x_n :: \tau_n\ \} \leq \{\ x'_1 :: \tau'_1, \ldots, x'_m :: \tau'_m\ \}}$$

Seems ok for both functional and imperative update...

Subtypes and Update

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                   r with (p = { x: 10 })):
  f({ p: { x: 5,
           y: 6 } }).p.y
```

Subtypes and Update

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                          r with (p = { x: 10 })):
  f({ p: { x: 5,
           y: 6 } }).p.y
```

```
{ p :: { x :: Int } }
```

Subtypes and Update

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                   r with (p = { x: 10 })):
  f({ p: { x: 5,
           y: 6 } }).p.y
```

```
{ p :: { x :: Int,
         y :: Int } }
```

vs.

```
{ p :: { x :: Int } }
```

Subtypes and Update

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                         r.p := { x: 10 }):
  let r :: .... = { p: { x: 5,
                         y: 6 } }:
    begin:
      f(r)
      r.p.y
```

Subtypes and Update

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                   r.p := { x: 10 }):
  let r :: .... = { p: { x: 5,
                         y: 6 } }:
    begin:
      f(r)
      r.p.y
```

```
{ p :: { x :: Int } }
```

Subtypes and Update

```
let f :: .... = (fun (r :: { p :: { x :: Int } }):
                   r.p := { x: 10 }):
  let r :: .... = { p: { x: 5,
                         y: 6 } }:
    begin:
      f(r)
      r.p.y
```

```
{ p :: { x :: Int,
         y :: Int } }
```

vs.

```
{ p :: { x :: Int } }
```

Subtypes and Update

```
{ p :: { x :: Int,
         y :: Int } }
```

vs.

```
{ p :: { x :: Int } }
```

$$\frac{\{x_1, \ldots, x_n\} \supseteq \{x'_1, \ldots, x'_m\} \quad x_i = x'_j \Rightarrow \tau_i \leq \tau'_j}{\{\ x_1 :: \tau_1, \ldots, x_n :: \tau_n\ \} \leq \{\ x'_1 :: \tau'_1, \ldots, x'_m :: \tau'_m\ \}}$$

Wrong for imperative update!

Subtypes and Update

```
{ p :: { x :: Int,
         y :: Int } }
```

vs.

```
{ p :: { x :: Int } }
```

$$\frac{\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \supseteq \{\mathbf{x}'_1, \ldots, \mathbf{x}'_m\} \qquad \mathbf{x}_i = \mathbf{x}'_j \Rightarrow \tau_i = \tau'_j}{\{\ \mathbf{x}_1 :: \tau_1, \ldots, \mathbf{x}_n :: \tau_n\ \} \leq \{\ \mathbf{x}'_1 :: \tau'_1, \ldots, \mathbf{x}'_m :: \tau'_m\ \}}$$

Invariance

With imperative update:

$$\frac{\{\mathbf{x}_1,\ \ldots,\ \mathbf{x}_n\} \supseteq \{\mathbf{x}'_1,\ \ldots,\ \mathbf{x}'_m\} \qquad \mathbf{x}_i = \mathbf{x}'_j \Rightarrow \tau_i = \tau'_j}{\texttt{\{}\ \mathbf{x}_1\ \texttt{::}\ \tau_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}_n\ \texttt{::}\ \tau_n\ \texttt{\}} \leq \texttt{\{}\ \mathbf{x}'_1\ \texttt{::}\ \tau'_1\texttt{,}\ \ldots\texttt{,}\ \mathbf{x}'_m\ \texttt{::}\ \tau'_m\ \texttt{\}}}$$

Field types must be ***invariant*** with record types