CSEP 505: Programming Languages

Lecture 9
March 5, 2015

$$\frac{x : \alpha \in \Gamma}{\Gamma \vdash x : \alpha} \quad \text{(VAR)}$$

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha . e) : \alpha \to \beta} \quad (\lambda)$$

$$\frac{\Gamma \vdash e_f : \alpha \to \beta \qquad \Gamma \vdash e_a : \alpha}{\Gamma \vdash (e_f \ e_a) : \beta} \quad \text{(APP)}$$

$$\frac{\Gamma \vdash e_1 : \alpha \qquad \Gamma \vdash e_2 : \beta}{\Gamma \vdash (\text{pair } e_1 \ e_2) : \alpha \times \beta} \quad \text{(PAIR)}$$

$$\frac{\Gamma \vdash e : \alpha \times \beta}{\Gamma \vdash (\text{fst } e) : \alpha} \quad \text{(FST)}$$

$$\frac{\Gamma \vdash e : \alpha \times \beta}{\Gamma \vdash (\text{snd } e) : \beta} \quad \text{(SND)}$$

```
data Bool = False | True

data Maybe a = Nothing | Just a

data Result a = Err String | Ok a
```

```
data Bool = False | True

data Maybe a = Nothing | Just a

data Result a = Err String | Ok a

data Either a b = Left a | Right b
```

```
type Bool = Either () ()

type Maybe a = Either () a

type Result a = Either String a

data Either a b = Left a | Right b
```

```
interp (VarE v) env =
  case lookup v env of
    Nothing → Err (“unbound: ” ++ v)
    Just val → Ok val
```

```
interp (VarE v) env =
  case lookup v env of
    Left () → Left ("unbound: " ++ v)
    Right val → Right val
```

```
(with* ([result (lookup var env)])
  (if (left? result)
      (left "unbound var")
      (right (unwrap result))))
```

```
(case (lookup var env)
  [_ => (left “unbound var”)]
  [val => (right val)])
```

```
(case (lookup var env)
  (fun (_) (left “unbound var”))
  (fun (val) (right val)))
```

```
(case (lookup var env)
  (fun (_) (left "unbound var"))
  (fun (val) (right val)))
```

$$\frac{\Gamma \vdash e : \alpha}{\Gamma \vdash (\text{left } e) : \alpha + \beta} \quad (\textsc{left})$$

$$\frac{\Gamma \vdash e : \beta}{\Gamma \vdash (\text{right } e) : \alpha + \beta} \quad (\textsc{right})$$

$$\frac{\Gamma \vdash e : \alpha + \beta \qquad \Gamma \vdash e_1 : \alpha \to \tau \qquad \Gamma \vdash e_2 : \beta \to \tau}{\Gamma \vdash (\text{case } e\ e_1\ e_2) : \tau} \quad (\textsc{case})$$

$$\Gamma \vdash \text{unit} : () \quad (\textsc{unit})$$

```
(+ 1 (call/cc
       (fun (k)
         (if (call/cc
               (fun (k')
                 (if …
                   (* 2 (k 0))
                   (k' true))))
             (+ 3 (k 4))
             1))))
```

$$\texttt{call/cc} :: \forall \texttt{ab}.((\texttt{a} \to \texttt{b}) \to \texttt{a}) \to \texttt{a}$$

```
data List a = Empty
            | Cons a (List a)
```

$$\text{List } a = \mu b.() + (a, b)$$

```
(fun ([x : μa.(a → ∀b.b)])
  (x x))
```

```
(fun ([x : μa.(a → ∀b.b)])
  (x x)) : (μa.(a → ∀b.b) → ∀b.b)
```

```
((fun ([x : μa.(a → ∀b.b)])
    (x x)) : (μa.(a → ∀b.b) → ∀b.b)
 (fun ([x : μa.(a → ∀b.b)])
    (x x))) : ∀b.b
```

```
(fun (f)
  ((fun (g) (f (g g))
   (fun (g) (f (g g))))
  : ∀a.((a → a) → a)
```

```
{ x : 3,
  y : 4 }
```

```
{ x : 3,
  y : 4 } : { x : num, y : num }
```

```
(fun ([pt : { x : num, y : num }])
  (+ (sqr pt.x)
     (sqr pt.y)))
```

```
((fun ([pt : { x : num, y : num }])
    (+ (sqr pt.x)
       (sqr pt.y)))
 { x : 3,
   y : 4 })
```

```
((fun ([pt : { x : num, y : num }])
    (+ (sqr pt.x)
       (sqr pt.y)))
 { x : 3,
   y : 4,
   name : "my point" })
```

$$\frac{\Gamma \vdash e : S \qquad S <: T}{\Gamma \vdash e : T} \qquad \text{(SUB)}$$

$$\frac{\Gamma \vdash e : S \qquad S <: T}{\Gamma \vdash e : T} \qquad (\textsc{sub})$$

$$T <: T \qquad (\textsc{refl})$$

$$\frac{\Gamma \vdash e : S \qquad S <: T}{\Gamma \vdash e : T} \qquad \text{(SUB)}$$

$$T <: T \qquad \text{(REFL)}$$

$$\{l_i : T_i \,^{i \in 1..n+k}\} <: \{l_i : T_i \,^{i \in 1..n}\} \qquad \text{(WIDTH)}$$

```
((fun ([pt : { x : num, y : num }])
   (+ (* pt.x pt.x)
      (* pt.y pt.y)))
 { x : 3,
   y : 4,
   name : "my point" })
```

```
{ center : { x : 3, y : 4 },
  radius : 5 }
```

```
{ center : { x : 3, y : 4 },
  radius : 5 }
 : { center : { x : num,
                y : num },
     radius : num }
```

```
(fun ([c : { center : { x : num,
                       y : num },
             radius : num }]
      [p : { x : num, y : num }])
  (< (+ (sqr (- p.x c.center.x))
        (sqr (- p.y c.center.y)))
     (sqr c.radius)))
```

```
((fun ([c : { center : { x : num,
                         y : num },
             radius : num }]
       [p : { x : num, y : num }])
  (< (+ (sqr (- p.x c.center.x))
        (sqr (- p.y c.center.y)))
     (sqr c.radius)))
 { center : { x : 3, y : 4 },
   radius : 5 }  { x : 1, y : 2 })
```

```
((fun ([c : { center : { x : num,
                         y : num },
             radius : num }]
       [p : { x : num, y : num }])
   (< (+ (sqr (- p.x c.center.x))
         (sqr (- p.y c.center.y)))
      (sqr c.radius)))
  { center : { x : 3, y : 4, z : 5 },
    radius : 5 }  { x : 1, y : 2 })
```

$$\frac{S_i <: T_i \text{ (for each } i \in 1..n)}{\{l_i : S_i{}^{i \in 1..n}\} <: \{l_i : T_i{}^{i \in 1..n}\}} \quad (\textsc{depth})$$

$$\frac{S_i <: T_i \text{ (for each } i \in 1..n)}{\{l_i : S_i{}^{i \in 1..n}\} <: \{l_i : T_i{}^{i \in 1..n}\}} \quad (\textsc{depth})$$

$$\frac{S <: U \qquad U <: T}{S <: T} \quad (\textsc{trans})$$

```
(fun ([f : {x : num, y : num} →
           {c : {x : num, y:num},
            r : num}]
      [lst : (list {x : num,
                    y : num})])
  (map f lst))
 : (list {c : {x : num, y:num},
          r : num})
```

```
((fun ([f : {x : num, y : num} →
            {c : {x : num, y:num},
             r : num}]
       [lst : (list {x : num,
                     y : num})])
   (map f lst))
 (fun ([p : {x : num, y : num}])
   {c : p, r : 10}))
```

```
((fun ([f : {x : num, y : num} →
            {c : {x : num, y:num},
             r : num}]
       [lst : (list {x : num,
                     y : num})])
  (map f lst))
 (fun ([p : {x : num, y : num}])
   {c : p, r : 10, color : "black"}))
```

```
((fun ([f : {x : num, y : num} →
            {c : {x : num, y:num},
             r : num}]
       [lst : (list {x : num,
                     y : num})])
   (map f lst))
 (fun ([p : {x: num, y: num, z: num}])
   {c : p, r : (* 2 p.z)}))
```

```
((fun ([f : {x : num, y : num} →
            {c : {x : num, y:num},
             r : num}]
        [lst : (list {x : num,
                      y : num})])
   (map f lst))
 (fun ([p : {x: num, y: num, z: num}])
   {c : p, r : (* 2 p.z)}))
```

```
((fun ([f : {x : num, y : num} →
            {c : {x : num, y:num},
             r : num}]
       [lst : (list {x : num,
                     y : num})])
  (map f lst))
 (fun ([p : {x : num}])
   {c : {x : p.x, y : (* 2 p.x)},
    r : 10}))
```

```
((fun ([f : {x : num, y : num} →
            {c : {x : num, y:num},
             r : num}]
       [lst : (list {x : num,
                     y : num})])
  (map f lst))
 (fun ([p : {}])
   {c : {x : 0, y : 0},
    r : 10}))
```

$$\frac{S_a <: T_a \qquad S_r <: T_r}{T_a \to S_r <: S_a \to T_r} \quad \text{(S-FUN)}$$

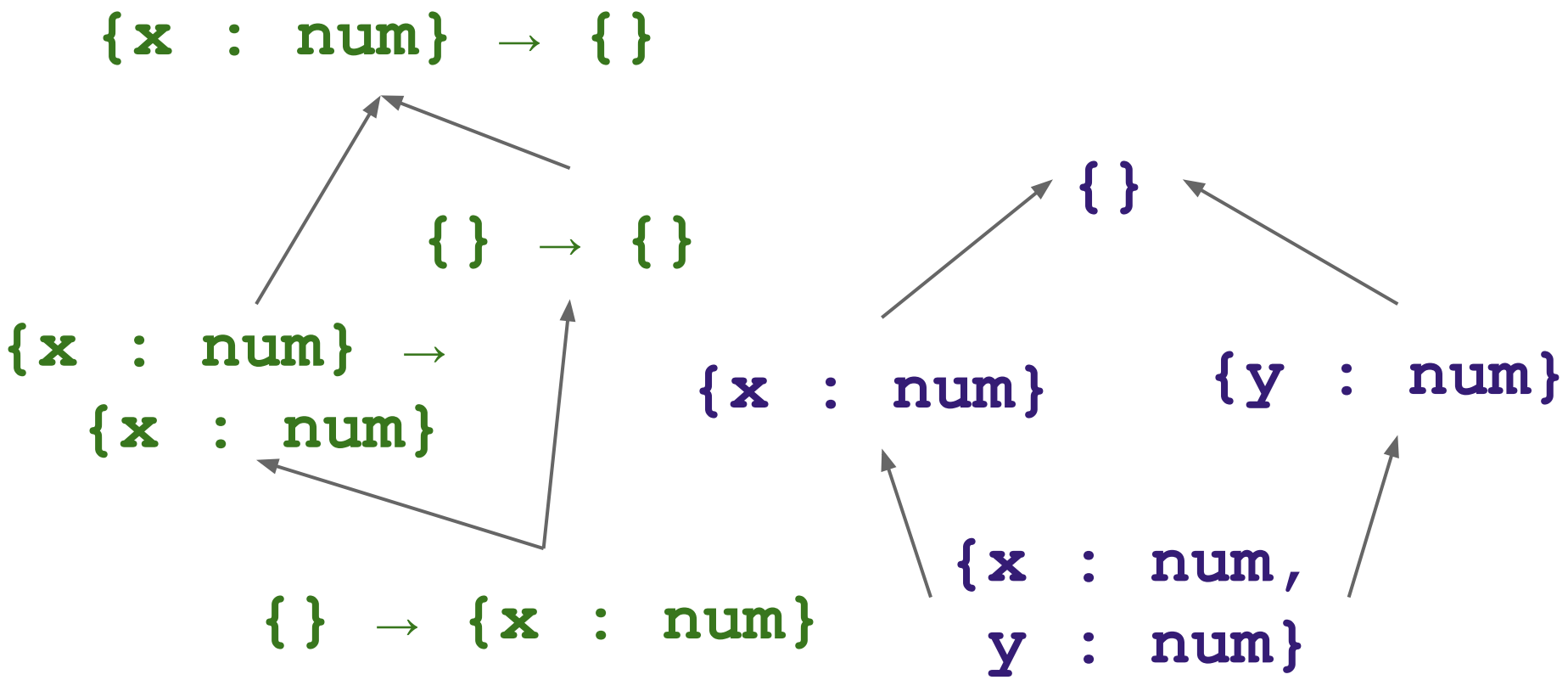
```
(with* (
    [g (fun ([b : (box {x : num})])
         (+ 1 (unbox b).x))]
    [a-box (box {x : 5})]
    [y (g a-box)]
  (if true
      (unbox a-box).x
      y))
```

```
(with* (
    [g (fun ([b : (box {x : num})])
         (+ 1 (unbox b).x))]
    [a-box (box {})]
    [y (g a-box)]
  (if true
      (unbox a-box).x
      y))
```

```
(with* (
    [g (fun ([b : (box {x : num})])
         (+ 1 (unbox b).x))]
    [a-box (box {x : 5, z : true})]
    [y (g a-box)]
  (if true
      (unbox a-box).x
      y))
```

```
(with* (
    [g (fun ([b : (box {x : num})])
         (set-box! b { x : 3 }))]
    [a-box (box {x : 5, z : true})]
    [y (g a-box)]
  (if true
      (unbox a-box).x
      y))
```

```
(with* (
    [g (fun ([b : (box {x : num})])
         (set-box! b { x : 3 }))]
    [a-box (box {x : 5, z : true})]
    [y (g a-box)]
  (if (unbox a-box).z
      (unbox a-box).x
      y))
```


{x : num} → {}
{} → {}
{x : num} →
{x : num}
{} → {x : num}
{}
{x : num}
{y : num}
{x : num,
y : num}

```
(if (zero? (f (+ n 1)))
    { x : (f n),
      y : 2,
      color : “red” }
    { x : n,
      y : 5,
      name : “my point” })
```

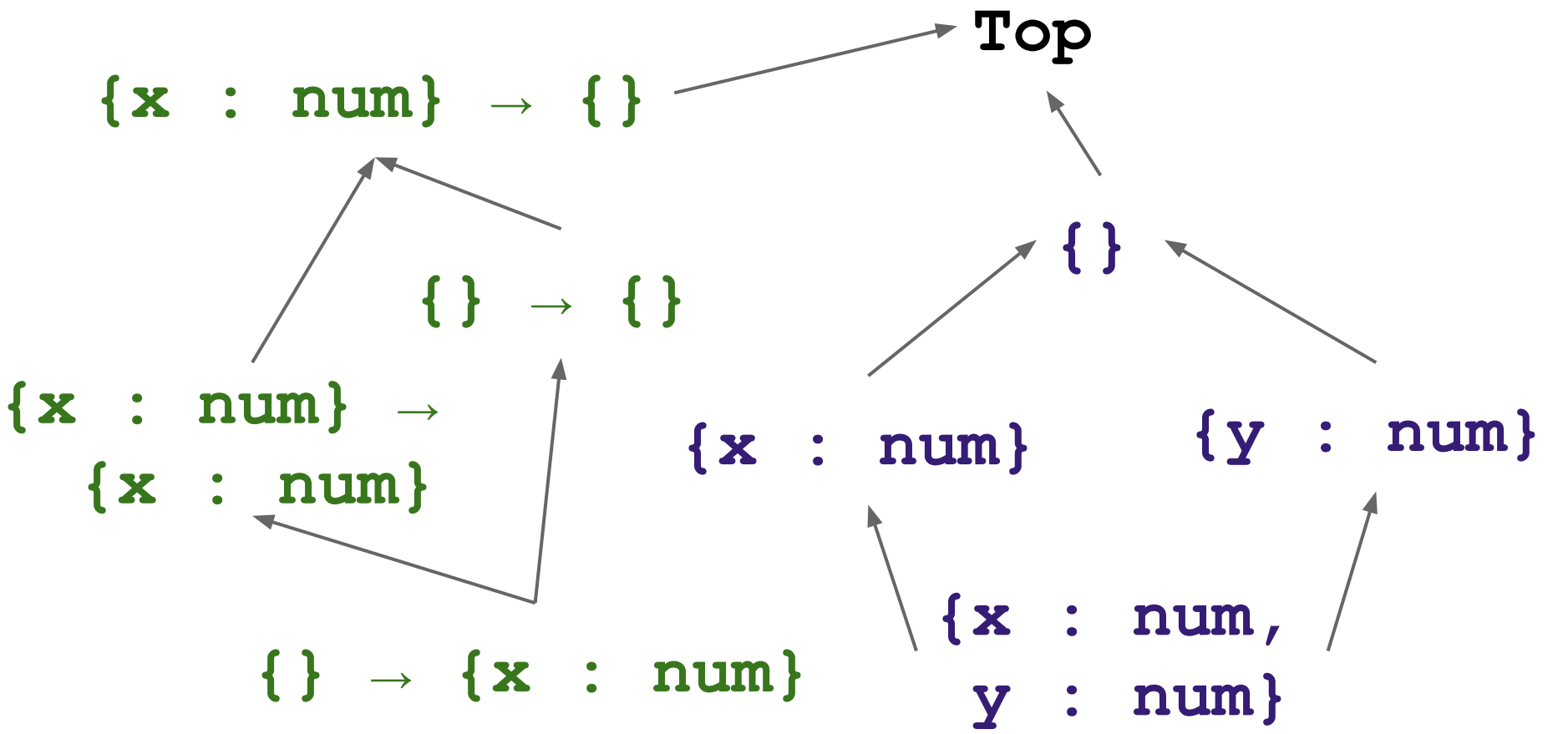
```
(if (zero? (f (+ n 1)))
    { x : (f n),

      color : “red” }
    { x : n,
      y : 5,
      name : “my point” })
```

```
(if (zero? (f (+ n 1)))
    { x : (f n),

      color : “red” }
    {
      y : 5,
      name : “my point” })
```

```
(if (zero? (f (+ n 1)))
    false
    (fun ([x : num]) x))
```


Top
{x : num} → {}
{}
{} → {}
{x : num} →
{x : num}
{x : num}
{y : num}
{x : num,
y : num}
{} → {x : num}

```
Hashable = { hash : () → num }
HashSet =
  { add : Hashable → (),
    contains : Hashable → bool,
    asList : () → (list Hashable)
  }
Point =
  { x : num, y : num, hash : () → num }
```

```
(with* ([mySet (makeHashSet)])
  (seq (mySet.add (makePoint 1 2))
       (mySet.add (makePoint 3 4))
       (mySet.add (makePoint -1 10))
       (mySet.asList ())))
```

```
Hashable = { hash : () → num }
HashSet = ∀a.
  { add : a → (),
    contains : a → bool,
    asList : () → (list a)
  }
Point =
  { x : num, y : num, hash : () → num }
```

v ::= n | *op* | **λ***x*.e e ::= v | **if** e e e | *x* | e e | let x = e in e | ∀α.e | e<t>

t ::= num | bool | t → t | [t] | t x t | α | ∀α.t

$$\frac{\Gamma, \alpha \vdash e : \tau}{\Gamma \vdash (\forall \alpha . e) : \forall \alpha . \tau} \quad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash \tau \qquad \Gamma \vdash e : \forall \alpha . \tau'}{\Gamma \vdash e\langle \tau \rangle : [\alpha \leftarrow \tau]\tau'} \quad \text{(T-SPEC)}$$

$$\frac{\Gamma \vdash \tau_a \qquad \Gamma, x : \tau_a \vdash e : \tau}{\Gamma \vdash (\lambda x : \tau_a . e) : \tau_a \rightarrow \tau} \quad (\lambda)$$

```
Comparable =
  { compare : Comparable → num }
SortedSet = ∀a <: Comparable.
 { add : a → (),
   remove : a → (),
   asList : () → (list a) }
Point =
 { x : num, y : num,
   compare : Comparable → num }
```

```
(with* ([my-x 3]
        [my-y 4])
  { x : my-x,
    y : my-y,
    compare :
      (fun ([other : Comparable])


        )})
```

```
Comparable =
   ∀a <: Comparable<a>
    { compare : a → num }
SortedSet = ∀a <: Comparable<a>.
 { add : a → (), remove : a → (),
   asList : () → (list a) }
Point =
 { x : num, y : num,
   compare : Point → num }
```

Concepts

- Curry-Howard correspondence
- Recursive types
- Subtyping
- Bounded quantification