

# Lecture 12 — Type System

## (3) Manual Type Annotation

CSE307: Programming Languages

Minseok Jeon

2026 Spring

# Typing Rules

---

$$\begin{array}{c} \overline{\Gamma \vdash n : \text{int}} \quad \overline{\Gamma \vdash x : \Gamma(x)} \\ \\ \frac{\Gamma \vdash E_1 : \text{int} \quad \Gamma \vdash E_2 : \text{int}}{\Gamma \vdash E_1 + E_2 : \text{int}} \quad \frac{\Gamma \vdash E_1 : \text{int} \quad \Gamma \vdash E_2 : \text{int}}{\Gamma \vdash E_1 - E_2 : \text{int}} \\ \\ \frac{\Gamma \vdash E : \text{int}}{\Gamma \vdash \text{iszero } E : \text{bool}} \quad \frac{\Gamma \vdash E_1 : \text{bool} \quad \Gamma \vdash E_2 : t \quad \Gamma \vdash E_3 : t}{\Gamma \vdash \text{if } E_1 \text{ then } E_2 \text{ else } E_3 : t} \\ \\ \frac{\Gamma \vdash E_1 : t_1 \quad [x \mapsto t_1] \Gamma \vdash E_2 : t_2}{\Gamma \vdash \text{let } x = E_1 \text{ in } E_2 : t_2} \quad \frac{\Gamma \vdash E_1 : t_1 \rightarrow t_2 \quad \Gamma \vdash E_2 : t_1}{\Gamma \vdash E_1 E_2 : t_2} \\ \\ \frac{[x \mapsto t_1] \Gamma \vdash E : t_2}{\Gamma \vdash \text{proc } x E : t_1 \rightarrow t_2} \end{array}$$

## Implementation: First Try

---

Can we implement the type checker recursively (like interpreter)?

```
let rec typeof  $\Gamma$   $E$  =  
  match  $E$  with  
  |  $n$   $\rightarrow$  int  
  |  $x$   $\rightarrow$   $\Gamma(x)$   
  |  $E_1 + E_2 \rightarrow$   
    let  $t_1 = \text{typeof } \Gamma E_1$  in  
    let  $t_2 = \text{typeof } \Gamma E_2$  in  
    if  $t_1 = \text{int}$  and  $t_2 = \text{int}$  then int  
    else raise TypeError  
     $\vdots$   
  | proc  $x$   $E \rightarrow$ 
```

## Challenge

---

Given a program  $E$ , how to check  $[\ ] \vdash E : t$ ? Nontrivial, because of the following type rule:

$$\frac{[x \mapsto t_1]\Gamma \vdash E : t_2}{\Gamma \vdash \text{proc } x \ E : t_1 \rightarrow t_2}$$

Two approaches:

- *Type Annotation*: Programmers are required to supply the type of the function argument. Used in C, C++, Java, etc.
- *Type Inference*: Type checker attempts to automatically infer types. Only possible if the language is carefully designed. Used in ML, Haskell, etc.

# Language with Type Annotation

---

Consider the language with (recursive) procedures:

$$\begin{array}{l} E \rightarrow n \\ | x \\ | E + E \\ | E - E \\ | \text{iszero } E \\ | \text{if } E \text{ then } E \text{ else } E \\ | \text{let } x = E \text{ in } E \\ | \text{proc } (x : t) E \\ | \text{letrec } t_1 f(x : t_2) = E \text{ in } E \\ | E E \end{array}$$

# Examples

---

- `proc (x:int) (x+1)`
- `letrec int double (x: int) =  
 if iszero x then 0 else (double (x-1)) +2  
in double 2`
- `proc (f: (bool -> int)) proc (n: int) (f (iszero n))`

# Typing Rules

---

$$\begin{array}{c} \overline{\Gamma \vdash n : \text{int}} \quad \overline{\Gamma \vdash x : \Gamma(x)} \\ \frac{\Gamma \vdash E_1 : \text{int} \quad \Gamma \vdash E_2 : \text{int}}{\Gamma \vdash E_1 + E_2 : \text{int}} \quad \frac{\Gamma \vdash E_1 : \text{int} \quad \Gamma \vdash E_2 : \text{int}}{\Gamma \vdash E_1 - E_2 : \text{int}} \\ \frac{\Gamma \vdash E : \text{int}}{\Gamma \vdash \text{iszero } E : \text{bool}} \quad \frac{\Gamma \vdash E_1 : \text{bool} \quad \Gamma \vdash E_2 : t \quad \Gamma \vdash E_3 : t}{\text{if } E_1 \text{ then } E_2 \text{ else } E_3 : t} \\ \frac{\Gamma \vdash E_1 : t_1 \quad [x \mapsto t_1] \Gamma \vdash E_2 : t_2}{\Gamma \vdash \text{let } x = E_1 \text{ in } E_2 : t_2} \quad \frac{\Gamma \vdash E_1 : t_1 \rightarrow t_2 \quad \Gamma \vdash E_2 : t_1}{\Gamma \vdash E_1 E_2 : t_2} \\ \frac{[x \mapsto t_1] \Gamma \vdash E : t_2}{\Gamma \vdash \text{proc } (x : t_1) E : t_1 \rightarrow t_2} \quad \frac{[x \mapsto t_2, f \mapsto (t_2 \rightarrow t_1)] \Gamma \vdash E_1 : t_1 \quad [f \mapsto (t_2 \rightarrow t_1)] \Gamma \vdash E_2 : t}{\Gamma \vdash \text{letrec } t_1 f(x : t_2) = E_1 \text{ in } E_2 : t} \end{array}$$

# Type Check Algorithm

---

Now we can implement the type checking algorithm recursively:

```
let rec typeof  $\Gamma$   $E$  =  
  match  $E$  with  
  | proc  $(x : t_1)$   $E_1 \rightarrow$   
    let  $t_2 = \mathbf{typeof} (\{x \mapsto t_1\}\Gamma) E_1$   
    in  $t_1 \rightarrow t_2$   
  | letrec  $t_1 f(x : t_2) = E_1$  in  $E_2$   
    let  $t' = \mathbf{typeof} (\{x \mapsto t_2, f \mapsto (t_2 \rightarrow t_1)\}\Gamma) E_1$   
    in if  $t' = t_1$  then typeof  $(\{f \mapsto (t_2 \rightarrow t_1)\}\Gamma) E_2$   
       else raise TypeError  
    :  
    :
```