COSE212: Programming Languages

Lecture 12 — Type System
(3) Manual Type Annotation

Hakjoo Oh
2022 Fall

Typing Rules

$$\overline{\Gamma \vdash n : \mathsf{int}} \qquad \overline{\Gamma \vdash x : \Gamma(x)}$$

$$\frac{\Gamma \vdash E_1 : \mathsf{int} \quad \Gamma \vdash E_2 : \mathsf{int}}{\Gamma \vdash E_1 + E_2 : \mathsf{int}} \qquad \frac{\Gamma \vdash E_1 : \mathsf{int} \quad \Gamma \vdash E_2 : \mathsf{int}}{\Gamma \vdash E_1 - E_2 : \mathsf{int}}$$

$$\frac{\Gamma \vdash E : \mathsf{int}}{\Gamma \vdash \mathtt{iszero}\ E : \mathsf{bool}} \qquad \frac{\Gamma \vdash E_1 : \mathsf{bool} \quad \Gamma \vdash E_2 : t \quad \Gamma \vdash E_3 : t}{\Gamma \vdash \mathtt{if}\ E_1\ \mathtt{then}\ E_2\ \mathtt{else}\ E_3 : t}$$

$$\frac{\Gamma \vdash E_1 : t_1 \quad [x \mapsto t_1]\Gamma \vdash E_2 : t_2}{\Gamma \vdash \mathtt{let}\ x = E_1\ \mathtt{in}\ E_2 : t_2} \qquad \frac{\Gamma \vdash E_1 : t_1 \to t_2 \quad \Gamma \vdash E_2 : t_1}{\Gamma \vdash E_1\ E_2 : t_2}$$

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

Implementation: First Try

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

```
let rec typeof Γ E =
  match E with
  | n → int
  | x → Γ(x)
  | E₁ + E₂ →
    let t₁ = typeof Γ E₁
    let t₂ = typeof Γ E₂
      if t₁ = int and t₂ = int then int
      else raise TypeError
      ⋮
  | proc x E →
```

Challenge

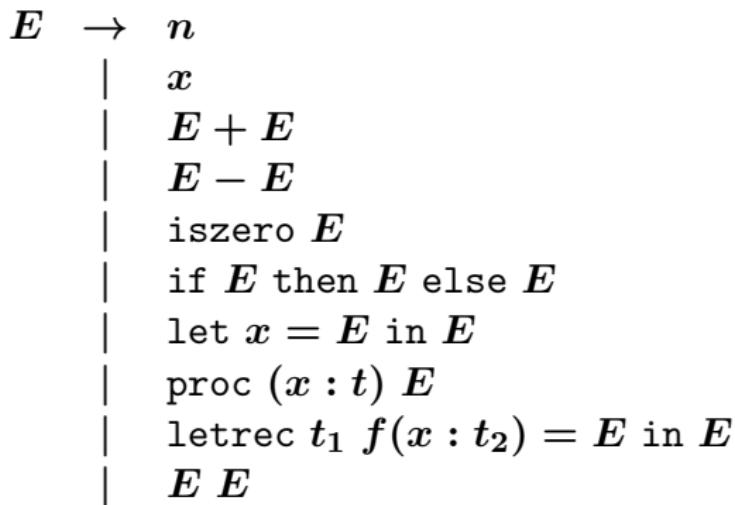Given a program E, how to check $[] \vdash E : t$? Nontrivial, because of the following type rule:

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

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:

$$
\begin{aligned}
E \;\rightarrow\; & n \\
\mid\; & x \\
\mid\; & E + E \\
\mid\; & E - E \\
\mid\; & \texttt{iszero } E \\
\mid\; & \texttt{if } E \texttt{ then } E \texttt{ else } E \\
\mid\; & \texttt{let } x = E \texttt{ in } E \\
\mid\; & \texttt{proc } (x : t)\ E \\
\mid\; & \texttt{letrec } t_1\ f(x : t_2) = E \texttt{ in } E \\
\mid\; & E\ E
\end{aligned}
$$

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

$$\overline{\Gamma \vdash n : \mathsf{int}} \qquad \overline{\Gamma \vdash x : \Gamma(x)}$$

$$\frac{\Gamma \vdash E_1 : \mathsf{int} \qquad \Gamma \vdash E_2 : \mathsf{int}}{\Gamma \vdash E_1 + E_2 : \mathsf{int}} \qquad \frac{\Gamma \vdash E_1 : \mathsf{int} \qquad \Gamma \vdash E_2 : \mathsf{int}}{\Gamma \vdash E_1 - E_2 : \mathsf{int}}$$

$$\frac{\Gamma \vdash E : \mathsf{int}}{\Gamma \vdash \mathtt{iszero}\ E : \mathsf{bool}} \qquad \frac{\Gamma \vdash E_1 : \mathsf{bool} \qquad \Gamma \vdash E_2 : t \qquad \Gamma \vdash E_3 : t}{\mathtt{if}\ E_1\ \mathtt{then}\ E_2\ \mathtt{else}\ E_3 : t}$$

$$\frac{\Gamma \vdash E_1 : t_1 \qquad [x \mapsto t_1]\Gamma \vdash E_2 : t_2}{\Gamma \vdash \mathtt{let}\ x = E_1\ \mathtt{in}\ E_2 : t_2} \qquad \frac{\Gamma \vdash E_1 : t_1 \to t_2 \qquad \Gamma \vdash E_2 : t_1}{\Gamma \vdash E_1\ E_2 : t_2}$$

$$\frac{[x \mapsto t_1]\Gamma \vdash E : t_2}{\Gamma \vdash \mathtt{proc}\ (x : t_1)\ E : t_1 \to t_2} \qquad \frac{\begin{array}{c} [x \mapsto t_2, f \mapsto (t_2 \to t_1)]\Gamma \vdash E_1 : t_1 \\ [f \mapsto (t_2 \to t_1)]\Gamma \vdash E_2 : t \end{array}}{\Gamma \vdash \mathtt{letrec}\ t_1\ f(x : t_2) = E_1\ \mathtt{in}\ E_2 : t}$$

Type Check Algorithm

Now we can implement the type checking algorithm recursively:

```
let rec typeof Γ E =
  match E with
  | proc (x : t₁) E₁ →
    let t₂ = typeof ({x ↦ t₁}Γ) E₁
    in t₁ → t₂
  | letrec t₁ f(x : t₂) = E₁ in E₂
    let t' = typeof ({x ↦ t₂, f ↦ (t₂ → t₁)}Γ) E₁
    in if t' = t₁ then typeof ({f ↦ (t₂ → t₁)}Γ) E₂
      else raise TypeError
      ⋮
```