# COSE212: Programming Languages

## Lecture 12 — Type System (3) Manual Type Annotation

Hakjoo Oh
2019 Fall

# Typing Rules

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

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

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

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

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

# Implementation: First Try

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

$$
\begin{array}{l}
\text{let rec } \textbf{typeof } \Gamma\ E = \\
\quad \text{match } E \text{ with} \\
\quad \mid n \rightarrow \text{int} \\
\quad \mid x \rightarrow \Gamma(x) \\
\quad \mid E_1 + E_2 \rightarrow \\
\qquad \text{let } t_1 = \textbf{typeof } \Gamma\ E_1 \\
\qquad \text{let } t_2 = \textbf{typeof } \Gamma\ E_2 \\
\qquad\quad \text{if } t_1 = \text{int and } t_2 = \text{int then int} \\
\qquad\quad \text{else raise TypeError} \\
\qquad\quad \vdots \\
\quad \mid \texttt{proc}\ x\ E \rightarrow
\end{array}
$$

# Challenge

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

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

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:

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

# 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

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

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

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

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

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

# Type Check Algorithm

Now we can implement the type checking algorithm recursively:

let rec **typeof** $\Gamma$ $E =$
  match $E$ with
  | $\texttt{proc}$ $(x : t_1)$ $E_1 \rightarrow$
    let $t_2 =$ **typeof** $(\{x \mapsto t_1\}\Gamma)$ $E_1$
    in $t_1 \rightarrow t_2$
  | $\texttt{letrec}$ $t_1$ $f(x : t_2) = E_1$ $\texttt{in}$ $E_2$
    let $t' =$ **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
      $\vdots$