ENE4014: Programming Languages

Lecture 11 — Type System
(2) Design

Woosuk Lee
2024 Spring

Language

$$
\begin{array}{rcl}
E & \to & n \\
& | & x \\
& | & E + E \\
& | & E - E \\
& | & \texttt{iszero}\ E \\
& | & \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
& | & \texttt{let}\ x = E\ \texttt{in}\ E \\
& | & \texttt{proc}\ x\ E \\
& | & E\ E
\end{array}
$$

Language

$$\frac{}{\rho \vdash n \Rightarrow n} \qquad \frac{}{\rho \vdash x \Rightarrow \rho(x)} \qquad \frac{\rho \vdash E_1 \Rightarrow n_1 \qquad \rho \vdash E_2 \Rightarrow n_2}{\rho \vdash E_1 + E_2 \Rightarrow n_1 + n_2}$$

$$\frac{\rho \vdash E \Rightarrow 0}{\rho \vdash \texttt{iszero}\ E \Rightarrow true} \qquad \frac{\rho \vdash E \Rightarrow n}{\rho \vdash \texttt{iszero}\ E \Rightarrow false}\ n \neq 0$$

$$\frac{\rho \vdash E_1 \Rightarrow true \qquad \rho \vdash E_2 \Rightarrow v}{\rho \vdash \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 \Rightarrow v} \qquad \frac{\rho \vdash E_1 \Rightarrow false \qquad \rho \vdash E_3 \Rightarrow v}{\rho \vdash \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 \Rightarrow v}$$

$$\frac{\rho \vdash E_1 \Rightarrow v_1 \qquad [x \mapsto v_1]\rho \vdash E_2 \Rightarrow v}{\rho \vdash \texttt{let}\ x = E_1\ \texttt{in}\ E_2 \Rightarrow v}$$

$$\frac{}{\rho \vdash \texttt{proc}\ x\ E \Rightarrow (x, E, \rho)}$$

$$\frac{\rho \vdash E_1 \Rightarrow (x, E, \rho') \qquad \rho \vdash E_2 \Rightarrow v \qquad [x \mapsto v]\rho' \vdash E \Rightarrow v'}{\rho \vdash E_1\ E_2 \Rightarrow v'}$$

Types

Types are defined inductively:

$$\begin{array}{rcl} \boldsymbol{T} & \rightarrow & \text{int} \\ & | & \text{bool} \\ & | & \boldsymbol{T} \rightarrow \boldsymbol{T} \end{array}$$

Examples:

- int
- bool
- int $\rightarrow$ int
- bool $\rightarrow$ int
- int $\rightarrow$ (int $\rightarrow$ bool)
- (int $\rightarrow$ int) $\rightarrow$ (bool $\rightarrow$ bool)
- (int $\rightarrow$ int) $\rightarrow$ (bool $\rightarrow$ (bool $\rightarrow$ int))

Types of Expressions

In order to compute the type of an expression, we need *type environment*:

$$\Gamma : Var \rightarrow T$$

Notation:

$\Gamma \vdash e : t \Leftrightarrow$ Under type environment Γ, expression e has type t.

Examples

- $[] \vdash \mathbf{3} : \text{int}$
- $[x \mapsto \text{int}] \vdash x : \text{int}$
- $[] \vdash \mathbf{4} - \mathbf{3} :$
- $[x \mapsto \text{int}] \vdash x - \mathbf{3} :$
- $[] \vdash \texttt{iszero}\ \mathbf{11} :$
- $[] \vdash \texttt{proc}\ (x)\ (x - \mathbf{11}) :$
- $[] \vdash \texttt{proc}\ (x)\ (\texttt{let}\ y = x - \mathbf{11}\ \texttt{in}\ (x - y)) :$
- $[] \vdash \texttt{proc}\ (x)\ (\texttt{if}\ x\ \texttt{then}\ \mathbf{11}\ \texttt{else}\ \mathbf{22}) :$
- $[] \vdash \texttt{proc}\ (x)\ (\texttt{proc}\ (y)\ \texttt{if}\ y\ \texttt{then}\ x\ \texttt{else}\ \mathbf{11}) :$
- $[] \vdash \texttt{proc}\ (f)\ (\texttt{if}\ (f\ \mathbf{3})\ \texttt{then}\ \mathbf{11}\ \texttt{else}\ \mathbf{22}) :$
- $[] \vdash (\texttt{proc}\ (x)\ x)\ \mathbf{1} :$
- $[f \mapsto \text{int} \to \text{int}] \vdash (f\ (f\ \mathbf{1})) :$

Typing Rules

Inductive rules for assigning types to expressions:

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

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

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

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

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

We say that a closed expression E has type t iff we can derive $[] \vdash E : t$.

Example 1

$$\frac{}{[] \vdash \texttt{iszero}\ (\mathbf{1} + \mathbf{2}) : \mathsf{bool}}$$

Example 2

$$\frac{}{[] \vdash \texttt{proc}\ (x)\ (x - 11) : \text{int} \to \text{int}}$$

Example 3

$$[] \vdash \texttt{proc}\ (x)\ (\texttt{if}\ x\ \texttt{then}\ \mathbf{11}\ \texttt{else}\ \mathbf{22}) : \text{bool} \to \text{int}$$

Example 4

$$\frac{}{[] \vdash (\texttt{proc}\ (x)\ x)\ \mathbf{1} : \mathsf{int}}$$

Example 5

$$[] \vdash \texttt{proc}\ (x)\ (\texttt{proc}\ (y)\ \texttt{if}\ y\ \texttt{then}\ x\ \texttt{else}\ \mathbf{11}) : \text{int} \to (\text{bool} \to \text{int})$$

Property 1 (Multiple Types)

Type assignment may not be unique:

- $\texttt{proc}\ x\ x$:

$$\frac{[x \mapsto \text{int}] \vdash x : \text{int}}{[] \vdash \texttt{proc}\ x\ x : \text{int} \to \text{int}}$$

$$\frac{[x \mapsto \text{bool}] \vdash x : \text{bool}}{[] \vdash \texttt{proc}\ x\ x : \text{bool} \to \text{bool}}$$

$$\frac{[x \mapsto (\text{int} \to \text{int})] \vdash x : \text{int} \to \text{int}}{[] \vdash \texttt{proc}\ x\ x : (\text{int} \to \text{int}) \to (\text{int} \to \text{int})}$$

- $\texttt{proc}\ (f)\ (f\ 3)$ has type $(\text{int} \to t) \to t$ for any t.
- The type of $\texttt{proc}\ (f)\ \texttt{proc}\ (x)\ (f\ (f\ x))$?

Property 2 (Soundness)

The type system is sound:

- If a closed expression $\boldsymbol{E}$ is well-typed

$$[] \vdash \boldsymbol{E} : \boldsymbol{t}$$

  for some $\boldsymbol{t} \in \boldsymbol{T}$, $\boldsymbol{E}$ does not have type error and produce a value:

$$[] \vdash \boldsymbol{E} \Rightarrow \boldsymbol{v}$$

- Furthermore, the type of $\boldsymbol{v}$ is $\boldsymbol{t}$. In other words, if $\boldsymbol{E}$ has a type error, we cannot find $\boldsymbol{t}$ such that $[] \vdash \boldsymbol{E} : \boldsymbol{t}$.
- Examples:
  - `(proc (x) x) 1`
  - `(proc (x) (x 3)) 4`

Property 2 (Soundness)

Theorem (Soundness of type system)

If $\boldsymbol{E}$ *is a closed expression,* $[] \vdash \boldsymbol{E} : \boldsymbol{t} \implies [] \vdash \boldsymbol{E} \Rightarrow \boldsymbol{v}$ *and* $\boldsymbol{v} : \boldsymbol{t}$

Proof.

- Case $\boldsymbol{E} = \boldsymbol{n}$: $[] \vdash \boldsymbol{E} :$ int, and $[] \vdash \boldsymbol{E} \Rightarrow \boldsymbol{n}$. Because $\boldsymbol{n} :$ int, the theorem holds.
- Case $\boldsymbol{E} = \boldsymbol{x}$: not a closed expression.
- Case $\boldsymbol{E} = \boldsymbol{E_1} + \boldsymbol{E_2}$: We will show $[] \vdash \boldsymbol{E} :$ int $\implies [] \vdash \boldsymbol{E} \Rightarrow \boldsymbol{n}$ for some integer $\boldsymbol{n}$.
  Inductive hypothesis (IH): $[] \vdash \boldsymbol{E_1} :$ int $\implies [] \vdash \boldsymbol{E_1} \Rightarrow \boldsymbol{n_1}$ and $[] \vdash \boldsymbol{E_2} :$ int $\implies [] \vdash \boldsymbol{E_2} \Rightarrow \boldsymbol{n_2}$ for some integers $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$.
  If $[] \vdash \boldsymbol{E_1} + \boldsymbol{E_2} :$ int, then $[] \vdash \boldsymbol{E_1} :$ int and $[] \vdash \boldsymbol{E_2} :$ int. By IH, $[] \vdash \boldsymbol{E_1} \Rightarrow \boldsymbol{n_1}$ and $[] \vdash \boldsymbol{E_2} \Rightarrow \boldsymbol{n_2}$. Therefore, $[] \vdash \boldsymbol{E_1} + \boldsymbol{E_2} \Rightarrow \boldsymbol{n_1} + \boldsymbol{n_2}$ by the evaluation rules.
- The other cases: exercise

Property 3 (Incompleteness)

The type system is incomplete: even though some programs do not have type errors, they do not have types according to the type system:

- `if iszero 1 then 11 else (iszero 22)`
- `(proc (f) (f f)) (proc x x)`

Property 3 (Incompleteness)

Why two branches should have the same type?

$$(\texttt{if } \boldsymbol{E} \texttt{ then } \mathbf{1} \texttt{ else } (\boldsymbol{true})) + \mathbf{1}$$

Can we know the value of $\boldsymbol{E}$ in advance?

- Case 1) $\boldsymbol{E}$ always evaluates to $\boldsymbol{true}$, so the program is OK!
- Case 2) $\boldsymbol{E}$ always evaluates to $\boldsymbol{false}$, so NOT OK!
- Case 3) $\boldsymbol{E}$ evaluate to $\boldsymbol{true}$ or $\boldsymbol{false}$, so NOT OK!
- Case 4) $\boldsymbol{E}$ never terminates, so OK!

We cannnot know the evaluation result of $\boldsymbol{E}$ without execution. If it's possible, we can solve the halting problem! (Why?)

Getting Results without Execution is Impossible

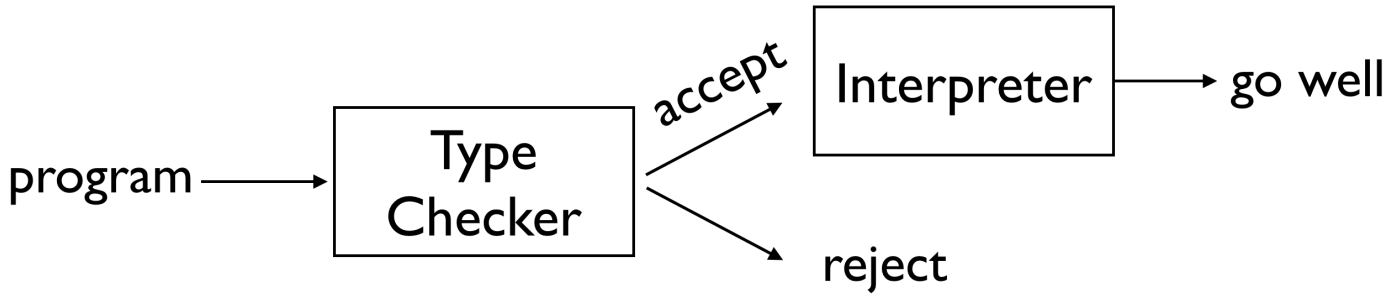
- Suppose we have an algorithm V that can exactly compute the value of a given expression p without execution.
- Then, we can construct $H(p)$ as follows:
  1. H takes p and construct the following program:

     $$p;\ \ true$$

  2. Invoke the procedure V with this program
     - If V returns $true$, then H returns yes (p terminates).
     - If V returns no value, then H returns no (p does not terminates).

Implementation

Implement a type checker according to the design:



- The type checker accepts a program E only if $[] \vdash E : t$ for some t.
- Otherwise, E is rejected.