COSE212: Programming Languages

Lecture 11 — Type System (2) Design

Hakjoo Oh
2025 Fall

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

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

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

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

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

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

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

Types

Types are defined inductively:

$$\begin{array}{rcl} T & \to & \mathsf{int} \\ & | & \mathsf{bool} \\ & | & T \to T \end{array}$$

Examples:

- int
- bool
- int $\to$ int
- bool $\to$ int
- int $\to$ (int $\to$ bool)
- (int $\to$ int) $\to$ (bool $\to$ bool)
- (int $\to$ int) $\to$ (bool $\to$ (bool $\to$ int))

Types of Expressions

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

$$\Gamma : Var \to T$$

Notation:

$$\Gamma \vdash e : t \Leftrightarrow \text{Under type environment } \Gamma\text{, expression } e \text{ has type } t.$$

Examples

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

Typing Rules

Inductive rules for assigning types to expressions:

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

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

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

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

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

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

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

Example 3

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

Example 4

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

Example 5

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

Property 1 (Multiple Types)

Type assignment may not be unique:

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

$$\frac{[x \mapsto \mathsf{int}] \vdash x : \mathsf{int}}{[] \vdash \mathtt{proc}\ x\ x : \mathsf{int} \to \mathsf{int}}$$

$$\frac{[x \mapsto \mathsf{bool}] \vdash x : \mathsf{bool}}{[] \vdash \mathtt{proc}\ x\ x : \mathsf{bool} \to \mathsf{bool}}$$

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

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

Property 2 (Soundness)

The type system is sound:

- If a closed expression E is well-typed

$$[] \vdash E : t$$

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

$$[] \vdash E \Rightarrow v$$

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

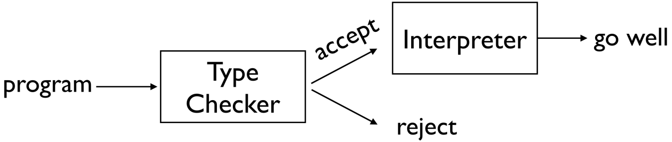
Property 3 (Incompleteness)

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

- $\texttt{if iszero } 1 \texttt{ then } 11 \texttt{ else } (\texttt{iszero } 22))$
- $(\texttt{proc } (f)\ (f\ f))\ (\texttt{proc } x\ x)$

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.

Type Soundness Proof

Theorem. If a closed expression E is well-typed, i.e.,

$$[] \vdash E : t,$$

then there exists a value v such that

$$[] \vdash E \Rightarrow v,$$

and moreover v has the canonical shape of t:

- if $t = \text{int}$, then v is an integer n;
- if $t = \text{bool}$, then $v \in \{true, false\}$;
- if $t = t_1 \to t_2$, then $v = (x, E', \rho')$ (closure).

Interpretation. Well-typed closed programs do not get stuck (no type errors) and evaluate to values matching their types.
(c.f., Termination guarantee is only for a language without recursive functions.)

Key Lemmas

Environment Consistency. We write $\rho \models \Gamma$ to mean: for every $x \in \mathbf{dom}(\Gamma)$, $\rho(x)$ is a value of type $\Gamma(x)$.

- *Extension:* If $\rho \models \Gamma$ and $v : t$, then $[x \mapsto v]\rho \models [x \mapsto t]\Gamma$.

Canonical Forms. If v is a value and $[] \vdash v : t$, then

- $t =$ int $\Rightarrow v = n$ for some integer n;
- $t =$ bool $\Rightarrow v \in \{true, false\}$;
- $t = t_1 \to t_2 \Rightarrow v = (x, E, \rho')$ (closure).

Generalized Theorem. For all Γ, E, t, ρ,

$$\Gamma \vdash E : t \ \wedge \ \rho \models \Gamma \quad \Longrightarrow \quad \exists v.\ \rho \vdash E \Rightarrow v \ \wedge \ v : t.$$

We then specialize to $\Gamma = []$ and $\rho = []$.

Proof Sketch

By induction on the derivation of $\Gamma \vdash E : t$.

- n, x: Trivial by evaluation rules and $\rho \models \Gamma$.
- $E_1 + E_2$, `iszero` E: From IH: $\rho \vdash E_i \Rightarrow n_i$; conclude by the evaluation rule.
- `if` E_1 `then` E_2 `else` E_3: IH on guard gives a boolean b; case-split and apply IH on the selected branch.
- `let` $x = E_1$ `in` E_2: IH on E_1 yields $v_1 : t_1$; extend environment via the extension lemma and apply IH to E_2.
- `proc` $x\ E'$: From typing $[x \mapsto t_1]\Gamma \vdash E' : t_2$; evaluation yields closure (x, E', ρ) which has type $t_1 \to t_2$.
- $E_1\ E_2$: IH gives $\rho \vdash E_1 \Rightarrow (x, E', \rho_c)$ and $\rho \vdash E_2 \Rightarrow v_2$ with $v_2 : t_1$; extend ρ_c and apply IH to E'.

Selected Cases (Details)

Case $\Gamma \vdash \texttt{if } E_1 \texttt{ then } E_2 \texttt{ else } E_3 : t$.

$$\Gamma \vdash E_1 : \text{bool}, \quad \Gamma \vdash E_2 : t, \quad \Gamma \vdash E_3 : t.$$

By IH, $\rho \vdash E_1 \Rightarrow b$ with $b \in \{true, false\}$.

- If $b = true$, IH on E_2 gives $\rho \vdash E_2 \Rightarrow v$ and $v : t$.
- If $b = false$, IH on E_3 gives $\rho \vdash E_3 \Rightarrow v$ and $v : t$.

By the evaluation rule, $\rho \vdash \texttt{if } E_1 \texttt{ then } E_2 \texttt{ else } E_3 \Rightarrow v$.

Case $\Gamma \vdash E_1\ E_2 : t_2$.

$$\Gamma \vdash E_1 : t_1 \to t_2, \qquad \Gamma \vdash E_2 : t_1.$$

- By IH, $\rho \vdash E_1 \Rightarrow (x, E', \rho_c)$, a closure, with typing guarantee $[x : t_1]\Gamma' \vdash E' : t_2$ for some Γ' and $\rho_c \models \Gamma'$.
- By IH, $\rho \vdash E_2 \Rightarrow v_2$ with $v_2 : t_1$.
- By the extension lemma, $[x \mapsto v_2]\rho_c \models [x : t_1]\Gamma'$.
- Applying IH to E', we get $[x \mapsto v_2]\rho_c \vdash E' \Rightarrow v$ and $v : t_2$.

Thus, by the application rule, $\rho \vdash E_1\ E_2 \Rightarrow v$.

Summary

- By induction on typing derivations and the lemmas, any $[] \vdash \boldsymbol{E} : \boldsymbol{t}$ evaluates: $[] \vdash \boldsymbol{E} \Rightarrow \boldsymbol{v}$.
- The resulting value $\boldsymbol{v}$ has the canonical form of $\boldsymbol{t}$ (no stuck states).
- Hence, the type system is **sound**.

Remark. Our language here has no general recursion; thus well-typed closed programs also terminate in this setting.