COSE212: Programming Languages

Lecture 5 — Design and Implementation of PLs (1) Expressions

Hakjoo Oh
2025 Fall

Plan

- **Part 1 (Preliminaries):** inductive definition, basics of functional programming, recursive and higher-order programming
- **Part 2 (Basic concepts):** syntax, semantics, naming, binding, scoping, environment, interpreters, states, side-effects, store, reference, mutable variables, parameter passing
- **Part 3 (Advanced concepts):** type system, typing rules, type checking, soundness/completeness, automatic type inference, polymorphic type system, lambda calculus, program synthesis

Goal

- We will learn essential concepts of programming languages by designing and implementing a programming language, called ML−−:
  - Expressions
  - Procedures
  - States
  - Types
- Design decisions of programming languages
  - Expression/statement-oriented
  - Static/dynamic scoping
  - Eager/lazy evaluation
  - Explicit/implicit reference
  - Static/dynamic type system
  - Sound/unsound type system
  - Manual/automatic type inference
  - ...

Designing a Programming Language

We need to specify syntax and semantics of the language:

- Syntax: how to write programs
- Semantics: the meaning of the programs

Both are formally specified by inductive definitions.

Let: Our First Language

Syntax

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

Examples

```
let x = 1 in x + 2
```

```
let x = 1
in let y = 2
   in x + y
```

```
let x = let y = 2
        in y + 1
in x + 3
```

```
let x = 1
in let y = 2
   in let x = 3
      in x + y
```

Examples

```
let x = 1
in let y = let x = 2
           in x + x
   in x + y
```

```
let x = 1
in let y = 2
   in if iszero (x - 1) then y - 1 else  y + 1
```

```
let x = 1
in let y = iszero x
   in x + y
```

Values and Environments

To define the semantics, we need to define **values** and **environments**.

- The set of values that the language manipulates:
  - `1+(2+3)`
  - `iszero 1, iszero (2-2)`
  - `if iszero 1 then 2 else 3`
- An environment is a variable-value mapping, which is needed to evaluate expressions with variables:
  - `x, y`
  - `x+1, x+(y-2)`
  - ```
    let x = read
    in let y = 2
       in if iszero x then y else x
    ```
    ```

Values and Environments

In Let, the set of values includes integers and booleans:

$$v \in Val = \mathbb{Z} + Bool$$

and an environment is a function from variables to values:

$$\rho \in Env = Var \rightarrow Val$$

Notations:

- []: the empty environment.
- $[x \mapsto v]\rho$ (or $\rho[x \mapsto v]$): the extension of ρ where x is bound to v:

$$([x \mapsto v]\rho)(y) = \begin{cases} v & \text{if } x = y \\ \rho(y) & \text{otherwise} \end{cases}$$

  For simplicity, we write $[x_1 \mapsto v_1, x_2 \mapsto v_2]\rho$ for the extension of ρ where x_1 is bound to v_1, x_2 to v_2:

$$[x_1 \mapsto v_1, x_2 \mapsto v_2]\rho = [x_1 \mapsto v_1]([x_2 \mapsto v_2]\rho)$$

Evaluation of Expressions

Given an environment ρ, an expression e evaluates to a value v:

$$\rho \vdash e \Rightarrow v$$

or does not evaluate to any value (i.e. e does not have semantics w.r.t ρ).

- $[] \vdash 1 \Rightarrow \mathbf{1}$
- $[x \mapsto \mathbf{1}] \vdash \texttt{x+1} \Rightarrow \mathbf{2}$
- $[] \vdash \texttt{read} \Rightarrow \mathbf{3}$, $[x \mapsto \mathbf{1}] \vdash \texttt{read} \Rightarrow \mathbf{5}$
- $[x \mapsto \mathbf{0}] \vdash \texttt{let y = 2 in if iszero x then y else x} \Rightarrow \mathbf{2}$
- `iszero (iszero 3)`
- `if 1 then 2 else 3`

Evaluation Rules

$$\boxed{\rho \vdash e \Rightarrow v}$$

$$\frac{}{\rho \vdash n \Rightarrow n} \qquad \frac{}{\rho \vdash x \Rightarrow \rho(x)}$$

$$\frac{\rho \vdash E_1 \Rightarrow n_1 \qquad \rho \vdash E_2 \Rightarrow n_2}{\rho \vdash E_1 + E_2 \Rightarrow n_1 + n_2} \qquad \frac{\rho \vdash E_1 \Rightarrow n_1 \qquad \rho \vdash E_2 \Rightarrow n_2}{\rho \vdash E_1 - E_2 \Rightarrow n_1 - n_2}$$

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

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

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

Evaluation Rules

More precise interpretation of the evaluation rules:

- The inference rules define a set S of triples (ρ, e, v). For readability, the triple was written by $\rho \vdash e \Rightarrow v$ in the rules.
- We say an expression e has semantics w.r.t. ρ iff there is a triple $(\rho, e, v) \in S$ for some value v.
- That is, we say an expression e has semantics w.r.t. ρ iff we can derive $\rho \vdash e \Rightarrow v$ for some value v by applying the inference rules.
- We say an initial program e has semantics if $[] \vdash e \Rightarrow v$ for some v.

Examples

$$\dfrac{\dfrac{}{[] \vdash 1 \Rightarrow \mathbf{1}} \quad \dfrac{\dfrac{}{[\mathrm{x} \mapsto \mathbf{1}] \vdash \mathrm{x} \Rightarrow \mathbf{1}} \quad \dfrac{}{[\mathrm{x} \mapsto \mathbf{1}] \vdash 2 \Rightarrow \mathbf{2}}}{[\mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{x + 2} \Rightarrow \mathbf{3}}}{[] \vdash \texttt{let x = 1 in x + 2} \Rightarrow \mathbf{3}}$$

Examples

$$\dfrac{[] \vdash \texttt{1} \Rightarrow \mathbf{1} \quad \dfrac{[\mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{2} \Rightarrow \mathbf{2} \quad \dfrac{\begin{array}{c}[\mathrm{y} \mapsto \mathbf{2}, \mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{x} \Rightarrow \mathbf{1} \\ [\mathrm{y} \mapsto \mathbf{2}, \mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{y} \Rightarrow \mathbf{2}\end{array}}{[\mathrm{y} \mapsto \mathbf{2}, \mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{x + y} \Rightarrow \mathbf{3}}}{[\mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{let y = 2 in x + y} \Rightarrow \mathbf{3}}}{[] \vdash \texttt{let x = 1 in let y = 2 in x + y} \Rightarrow \mathbf{3}}$$

Examples

$$\dfrac{\dfrac{[\,] \vdash 2 \Rightarrow \mathbf{2} \quad [\mathrm{y} \mapsto \mathbf{2}] \vdash \texttt{y + 1} \Rightarrow \mathbf{3}}{[\,] \vdash \texttt{let y = 2 in y + 1} \Rightarrow \mathbf{3}} \quad \dfrac{[\mathrm{x} \mapsto \mathbf{3}] \vdash \mathrm{x} \Rightarrow \mathbf{3} \quad [\mathrm{x} \mapsto \mathbf{3}] \vdash 3 \Rightarrow \mathbf{3}}{[\mathrm{x} \mapsto \mathbf{3}] \vdash \texttt{x + 3} \Rightarrow \mathbf{6}}}{[\,] \vdash \texttt{let x = (let y = 2 in y + 1) in x + 3} \Rightarrow \mathbf{6}}$$

Examples

$$[\mathrm{y} \mapsto \mathbf{2}, \mathrm{x} \mapsto \mathbf{3}] \vdash$$

$$[\mathrm{y} \mapsto \mathbf{2}, \mathrm{x} \mapsto \mathbf{3}] \vdash$$

$$[\mathrm{y} \mapsto \mathbf{2}, \mathrm{x} \mapsto \mathbf{1}] \vdash 3 \Rightarrow \mathbf{3} \quad [\mathrm{y} \mapsto \mathbf{2}, \mathrm{x} \mapsto \mathbf{3}] \vdash$$

$$[\mathrm{x} \mapsto \mathbf{1}] \vdash 2 \Rightarrow \mathbf{2} \quad [\mathrm{y} \mapsto \mathbf{2}, \mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{let x = 3 in x + y}$$

$$[] \vdash 1 \Rightarrow \mathbf{1} \quad [\mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{let y = 2 in let x = 3 in x + y} \Rightarrow \mathbf{5}$$

$$[] \vdash \texttt{let x = 1 in let y = 2 in let x = 3 in x + y} \Rightarrow \mathbf{5}$$

Examples

$$
\dfrac{
  \dfrac{
    \dfrac{[\mathrm{x} \mapsto \mathbf{2}] \vdash \texttt{x} \Rightarrow \mathbf{2} \quad [\mathrm{x} \mapsto \mathbf{2}] \vdash \texttt{x} \Rightarrow \mathbf{2}}{[\mathrm{x} \mapsto \mathbf{2}] \vdash \texttt{x + x} \Rightarrow \mathbf{4}}
  }{[\mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{let x = 2 in x + x} \Rightarrow \mathbf{4}}
  \quad
  \dfrac{\begin{array}{c}[\mathrm{y} \mapsto \mathbf{4}, \mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{x} \Rightarrow \mathbf{1} \\ [\mathrm{y} \mapsto \mathbf{4}, \mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{y} \Rightarrow \mathbf{4}\end{array}}{[\mathrm{y} \mapsto \mathbf{4}, \mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{x + y} \Rightarrow \mathbf{5}}
}{
  \dfrac{[\mathrm{x} \mapsto \mathbf{1}] \vdash \texttt{let y = (let x = 2 in x + x) in x + y} \Rightarrow \mathbf{5}}{[] \vdash \texttt{let x = 1 in let y = (let x = 2 in x + x) in x + y} \Rightarrow \mathbf{5}}
}
$$

Examples

When ρ is $[\mathrm{x} \mapsto \mathbf{1}, \mathrm{y} \mapsto \mathbf{2}]$:

$$\dfrac{\dfrac{\dfrac{\rho \vdash \texttt{x} \Rightarrow \mathbf{1} \quad \rho \vdash \texttt{1} \Rightarrow \mathbf{1}}{\rho \vdash \texttt{x - 1} \Rightarrow \mathbf{0}}}{\rho \vdash \texttt{iszero (x - 1)} \Rightarrow \boldsymbol{true}} \quad \dfrac{\rho \vdash \texttt{y} \Rightarrow \mathbf{2} \quad \rho \vdash \texttt{1} \Rightarrow \mathbf{1}}{\rho \vdash \texttt{y - 1} \Rightarrow \mathbf{1}}}{\rho \vdash \texttt{if iszero (x - 1) then y - 1 else y + 1} \Rightarrow \mathbf{1}}$$

Implementation of the Language

Syntax definition in OCaml:

```
type program = exp
and exp =
  | CONST of int
  | VAR of var
  | ADD of exp * exp
  | SUB of exp * exp
  | READ
  | ISZERO of exp
  | IF of exp * exp * exp
  | LET of var * exp * exp
and var = string
```

Example

```
let x = 7
in let y = 2
   in let  y = let x = x - 1
               in x - y
      in (x-8)-y
```

```
LET ("x", CONST 7,
   LET ("y", CONST 2,
      LET ("y", LET ("x", SUB(VAR "x", CONST 1),
                  SUB (VAR "x", VAR "y")),
        SUB (SUB (VAR "x", CONST 8), VAR "y"))))
```

Values and Environments

Values:

```
type value = Int of int | Bool of bool
```

Environments:

```
type env = (var * value) list
let empty_env = []
let extend_env (x,v) e = (x,v)::e
let rec lookup_env x e =
  match e with
  | [] -> raise (Failure ("variable " ^ x ^ " not found"))
  | (y,v)::tl -> if x = y then v else lookup_env x tl
```

Evaluation Rules

```
let rec eval : exp -> env -> value
=fun exp env ->
  match exp with
  | CONST n -> Int n
  | VAR x -> lookup_env x env
  | ADD (e1,e2) ->
    let v1 = eval e1 env in
    let v2 = eval e2 env in
      (match v1,v2 with
      | Int n1, Int n2 -> Int (n1 + n2)
      | _ -> raise (Failure "Type Error: non-numeric values"))
  | SUB (e1,e2) ->
    let v1 = eval e1 env in
    let v2 = eval e2 env in
      (match v1,v2 with
      | Int n1, Int n2 -> Int (n1 - n2)
      | _ -> raise (Failure "Type Error: non-numeric values"))
  ...
```

Implementation: Semantics

```
let rec eval : exp -> env -> value
=fun exp env ->
  ...
  | READ -> Int (read_int())
  | ISZERO e ->
    (match eval e env with
    | Int n when n = 0 -> Bool true
    | _ -> Bool false)
  | IF (e1,e2,e3) ->
    (match eval e1 env with
    | Bool true -> eval e2 env
    | Bool false -> eval e3 env
    | _ -> raise (Failure "Type Error: condition must be Bool type")
  | LET (x,e1,e2) ->
    let v1 = eval e1 env in
      eval e2 (extend_env (x,v1) env)
```

Interpreter

```
let run : program -> value
=fun pgm -> eval pgm empty_env
```

Examples:

```
# let e1 = LET ("x", CONST 1, ADD (VAR "x", CONST 2));;
val e1 : exp = LET ("x", CONST 1, ADD (VAR "x", CONST 2))
# run e1;;
- : value = Int 3
```

Summary

We have designed and implemented our first programming language:

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

- key concepts: syntax, semantics, interpreter