COSE212: Programming Languages

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

Hakjoo Oh
2016 Fall

Plan

- **Part 1 (Preliminaries):** inductive definition, basics of OCaml 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, type inference, polymorphism, modules, module procedures, typed modules, objects, classes, methods, inheritance, typed object-oriented languages

Overview

We will learn essential concepts of programming languages by designing and implementing a programming language in a incremental way:

- Expressions
- States
- Types
- Modules

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 (inference rules).

Our First Language

Syntax

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

Examples

- `1, 2, x, y`
- `1+(2+3), x+1, x+(y-2)`
- `zero? 1, zero? (2-2), zero? (zero? 3)`
- `if zero 1 then 2 else 3, if 1 then 2 else 3`
- ```
  let x = read
  in  x + 1
  ```
- ```
  let x = read
  in let y = 2
     in if zero x then y else x
  ```

Values and Environments

To define the semantics, we define values and environments.

- The set of values that the language manipulates, e.g., in Let,

$$\boldsymbol{Val} = \mathbb{Z} + \boldsymbol{Bool}$$

- Environments maintains variable bindings:

$$\boldsymbol{Env} = \boldsymbol{Var} \rightarrow \boldsymbol{Val}$$

  Notations:
  - ρ ranges over environments, i.e., $\rho \in \boldsymbol{Env}$.
  - $[]$: the empty environment.
  - $[x \mapsto v]\rho$: the extension of ρ where x is bound to v:

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

  - $[x_1 \mapsto v_1, x_2 \mapsto v_2]\rho$: the extension of ρ where x_1 is bound to v_1, x_2 to v_2:

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

Semantics

The notation

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

means that e evaluates to v in environment ρ.

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

Semantics

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

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

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

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

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

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

A program e has semantics w.r.t. ρ iff we can derive $\rho \vdash e \Rightarrow v$ for some value v by the inference rules.

Arithmetic Expressions

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

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

- When $\rho = [i \mapsto 1, v \mapsto 5, x \mapsto 10]$,

$$\frac{\dfrac{\dfrac{}{\rho \vdash x \Rightarrow 10} \qquad \dfrac{}{\rho \vdash 3 \Rightarrow 3}}{\rho \vdash x - 3 \Rightarrow 7} \qquad \dfrac{\dfrac{}{\rho \vdash v \Rightarrow 5} \qquad \dfrac{}{\rho \vdash i \Rightarrow 1}}{\rho \vdash v - i \Rightarrow 4}}{\rho \vdash (x - 3) - (v - i) \Rightarrow 3}$$

Arithmetic Expressions

- But expression $\boldsymbol{y - 3}$ does not have semantics because

$$\rho \vdash y - 3 \Rightarrow v$$

  cannot be derived for any value v.

- In $\rho = [x \mapsto true]$, the semantics of $x + 1$ is not defined because

$$\rho \vdash x + 1 \Rightarrow v$$

  cannot be derived for any v.

Conditional Expressions

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

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

- When $\rho = [x \mapsto 33, y \mapsto 22]$,

$$\frac{\dfrac{\dfrac{\overline{\rho \vdash x \Rightarrow 33} \qquad \overline{\rho \vdash 11 \Rightarrow 11}}{\rho \vdash x - 11 \Rightarrow 22}}{\rho \vdash \texttt{zero?}\ (x - 11) \Rightarrow \mathit{false}} \qquad \dfrac{\overline{\rho \vdash y \Rightarrow 22} \qquad \overline{\rho \vdash 4 \Rightarrow 4}}{\rho \vdash y - 4 \Rightarrow 18}}{\rho \vdash \texttt{if zero?}\ (x - 11)\ \texttt{then}\ y - 2\ \texttt{else}\ y - 4 \Rightarrow 18}$$

Let Expression

A let expression creates a new *variable binding* in the environment:

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

Example:

$$\frac{[] \vdash 5 \Rightarrow 5 \qquad \dfrac{\overline{[x \mapsto 5] \vdash x \Rightarrow 5} \qquad \overline{[x \mapsto 5] \vdash 3 \Rightarrow 3}}{[x \mapsto 5] \vdash x - 3 \Rightarrow 2}}{[] \vdash \texttt{let}\ x = 5\ \texttt{in}\ x - 3 \Rightarrow 2}$$

Let Expression

Let expressions can be nested:

- ```
  let z = 5
  in let x = 3
     in let y = x - 1
        in let x = 4
             in z - (x-y)
  ```
- ```
  let x = 7
  in let y = 2
     in let  y = let x = x - 1
                 in x - y
        in (x-8)-y
  ```

Implementing an Interpreter

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"))))
```

Implementation: Values and Environments

Values:

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

Environments:

```
type env = var -> value
let extend_env (x,v) e = fun y -> if x = y then v else (e y)
let apply_env e x = e x
let empty_env _ = raise (Failure "Env is empty")
```

Implementation: Semantics

```
let rec eval : exp -> env -> value
=fun exp env ->
  match exp with
  | CONST n -> Int n
  | VAR x -> apply_env env x
  | 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)
```

Example

Running the program:

```
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 a simple programming language:

$$
\begin{array}{lcl}
P & \rightarrow & E \\
E & \rightarrow & n \\
 & \mid & x \\
 & \mid & E + E \\
 & \mid & E - E \\
 & \mid & \texttt{zero?}\ E \\
 & \mid & \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
 & \mid & \texttt{let}\ x = E\ \texttt{in}\ E
\end{array}
$$

- key concepts: syntax, semantics, interpreter