

COSE212: Programming Languages

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

Hakjoo Oh
2017 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

$$\begin{array}{l} P \rightarrow E \\ E \rightarrow n \\ \quad | \\ \quad x \\ \quad | \\ \quad E + E \\ \quad | \\ \quad E - E \\ \quad | \\ \quad \text{iszero } E \\ \quad | \\ \quad \text{if } E \text{ then } E \text{ else } E \\ \quad | \\ \quad \text{let } x = E \text{ in } E \\ \quad | \\ \quad \text{read} \end{array}$$

Examples

- 1, 2, x, y
- $1+(2+3)$, $x+1$, $x+(y-2)$
- `iszero 1`, `iszero (2-2)`, `iszero (iszero 3)`
- `if iszero 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 need to define **values** and **environments**.

- The set of values that the language manipulates:
 - ▶ `1+(2+3)`
 - ▶ `iszero 1, iszero (2-2)`
 - ▶ `if zero 1 then 2 else 3`

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 zero 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 zero x then y else x`

Values and Environments

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

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

and an environment is a function from variables to values:

$$\rho \in \mathit{Env} = \mathit{Var} \rightarrow \mathit{Val}$$

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).

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

Evaluation Rules

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

$$\overline{\rho \vdash n \Rightarrow n} \quad \overline{\rho \vdash x \Rightarrow \rho(x)}$$

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

$$\overline{\rho \vdash \text{read} \Rightarrow n} \quad \frac{\rho \vdash E \Rightarrow 0}{\rho \vdash \text{iszero } E \Rightarrow \text{true}} \quad \frac{\rho \vdash E \Rightarrow n}{\rho \vdash \text{iszero } E \Rightarrow \text{false}} \quad n \neq 0$$

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

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

Evaluation Rules

More precise interpretation of the evaluation rules:

- The inference rules define a set \mathcal{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 $(\rho, e, v) \in \mathcal{S}$ 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: Arithmetic Expressions

$$\overline{\rho \vdash n \Rightarrow n} \quad \overline{\rho \vdash x \Rightarrow \rho(x)}$$

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

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

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

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

Examples: Arithmetic Expressions

- Expression $y - 3$ does not have semantics w.r.t the same ρ because

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

cannot be derived for any value v .

Examples: Arithmetic Expressions

- Expression $y - 3$ does not have semantics w.r.t the same ρ because

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

cannot be derived for any value v .

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

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

cannot be derived for any v .

Examples: Conditional Expressions

$$\frac{\rho \vdash E \Rightarrow 0}{\rho \vdash \text{iszero } E \Rightarrow \text{true}} \quad \frac{\rho \vdash E \Rightarrow n}{\rho \vdash \text{iszero } E \Rightarrow \text{false}} \quad n \neq 0$$

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

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

$$\frac{\frac{\overline{\rho \vdash x \Rightarrow 33} \quad \overline{\rho \vdash 11 \Rightarrow 11}}{\rho \vdash x - 11 \Rightarrow 22}}{\rho \vdash \text{iszero } (x - 11) \Rightarrow \text{false}} \quad \frac{\overline{\rho \vdash y \Rightarrow 22} \quad \overline{\rho \vdash 4 \Rightarrow 4}}{\rho \vdash y - 4 \Rightarrow 18}}{\rho \vdash \text{if iszero } (x - 11) \text{ then } y - 2 \text{ else } y - 4 \Rightarrow 18}$$

Examples: Let Expression

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

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

Example:

$$\frac{\begin{array}{c} \overline{[x \mapsto 5] \vdash x \Rightarrow 5} \quad \overline{[x \mapsto 5] \vdash 3 \Rightarrow 3} \\ [x \mapsto 5] \vdash x - 3 \Rightarrow 2 \end{array}}{\begin{array}{c} [] \vdash 5 \Rightarrow 5 \\ \hline [] \vdash \text{let } x = 5 \text{ in } x - 3 \Rightarrow 2 \end{array}}$$

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 x = x - 1`
 `in x - y`
 `in (x-8)-y`

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
```

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 apply_env x e =
  match e with
  | [] -> raise (Failure ("variable " ^ x ^ " not found"))
  | (y,v)::tl -> if x = y then v else apply_env x tl
```

Evaluation Rules

```
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)
```


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:

$$\begin{array}{l} P \rightarrow E \\ E \rightarrow n \\ \quad | x \\ \quad | E + E \\ \quad | E - E \\ \quad | \text{iszero } E \\ \quad | \text{if } E \text{ then } E \text{ else } E \\ \quad | \text{let } x = E \text{ in } E \end{array}$$

- key concepts: syntax, semantics, interpreter