COSE212: Programming Languages

Lecture 6 — Design and Implementation of PLs (2) Procedures

Hakjoo Oh

2023 Fall

Review: The Let Language

Syntax:

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

Review: The Let Language

Semantic domain:

$$\begin{array}{rcl} \mathit{Val} & = & \mathbb{Z} + \mathit{Bool} \\ \mathit{Env} & = & \mathit{Var} \rightarrow \mathit{Val} \end{array}$$

Semantics rules:

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

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

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

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

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

Proc = Let + Procedures

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

Example

- ```
  let f = proc (x) (x-11)
  in (f (f 77))
  ```
- `((proc (f) (f (f 77))) (proc (x) (x-11)))`
  ```

Free/Bound Variables of Procedures

- An occurrence of the variable `x` is *bound* when it occurs without definitions in the body of a procedure whose formal parameter is `x`.
- Otherwise, the variable is *free*.
- Examples:
  - `proc (y) (x+y)`
  - `proc (x) (let y = 1 in x + y + z)`
  - `proc (x) (proc (y) (x+y))`
  - `let x = 1 in proc (y) (x+y)`
  - `let x = 1 in proc (y) (x+y+z)`

Static vs. Dynamic Scoping

What is the result of the program?

```
let x = 1
in let f = proc (y) (x+y)
   in let x = 2
      in let g = proc (y) (x+y)
         in (f 1) + (g 1)
```

Two ways to determine free variables of procedures:

- In *static scoping* (*lexical scoping*), the procedure body is evaluated in the environment where the procedure is defined (i.e. procedure-creation environment).
- In *dynamic scoping*, the procedure body is evaluated in the environment where the procedure is called (i.e. calling environment)

Exercises

What is the result of the program?

- In static scoping:
- In dynamic scoping:

1. ```
   let a = 3
   in let p = proc (z) a
      in let f = proc (x) (p 0)
         in let a = 5
            in (f 2)
   ```
2. ```
   let a = 3
   in let p = proc (z) a
      in let f = proc (a) (p 0)
         in let a = 5
            in (f 2)
   ```

Why Static Scoping?

Most modern languages use static scoping. Why?

- Reasoning about programs is much simpler in static scoping.
- In static scoping, renaming bound variables by their lexical definitions does not change the semantics, which is unsafe in dynamic scoping.

```
let x = 1
in let f = proc (y) (x+y)
   in let x = 2
      in let g = proc (y) (x+y)
         in (f 1) + (g 1)
```

- In static scoping, names are resolved at compile-time.
- In dynamic scoping, names are resolved only at runtime.

Semantics of Procedures: Static Scoping

- Domain:

$$\begin{array}{rcl} Val & = & \mathbb{Z} + Bool + Procedure \\ Procedure & = & Var \times E \times Env \\ Env & = & Var \to Val \end{array}$$

  The procedure value is called *closures*. The procedure is closed in its creation environment.

- Semantics rules:

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

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

Examples

$$\frac{}{[\,] \vdash \texttt{(proc (x) (x)) 1} \Rightarrow \mathbf{1}}$$

Examples

$$\dfrac{}{[\,] \vdash \begin{array}{l}\texttt{let x = 1} \\ \texttt{in let f = proc (y) (x+y)} \\ \quad\texttt{in let x = 2} \\ \qquad\texttt{in (f 3)}\end{array} \Rightarrow \mathbf{4}}$$

Semantics of Procedures: Dynamic Scoping

- Domain:

$$\begin{array}{rcl} \boldsymbol{Val} & = & \mathbb{Z} + \boldsymbol{Bool} + \boldsymbol{Procedure} \\ \boldsymbol{Procedure} & = & \boldsymbol{Var} \times \boldsymbol{E} \\ \boldsymbol{Env} & = & \boldsymbol{Var} \to \boldsymbol{Val} \end{array}$$

- Semantics rules:

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

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

Examples

$$\frac{}{[\,] \vdash \begin{array}{l}\texttt{let x = 1}\\ \texttt{in let f = proc (y) (x+y)}\\ \quad\texttt{in let x = 2}\\ \qquad\texttt{in (f 3)}\end{array} \Rightarrow \mathbf{5}}$$

cf) Multiple Argument Procedures

- We can get the effect of multiple argument procedures by using procedures that return other procedures.
- ex) a function that takes two arguments and return their sum:

```
let f = proc (x) proc (y) (x+y)
in ((f 3) 4)
```

Adding Recursive Procedures

The current language does not support recursive procedures, e.g.,

```
let f = proc (x) (f x)
in (f 1)
```

for which evaluation gets stuck:

$$\dfrac{[f \mapsto (x, f\ x, [])] \vdash f \Rightarrow (x, f\ x, []) \qquad \dfrac{[x \mapsto 1] \vdash f \Rightarrow ? \qquad [x \mapsto 1] \vdash x \Rightarrow 1}{[x \mapsto 1] \vdash f\ x \Rightarrow ?}}{[f \mapsto (x, f\ x, [])] \vdash (f\ 1) \Rightarrow ?}$$

Two solutions:

- go back to dynamic scoping :-(
- modify the language syntax and semantics for procedure :-)

Recursion is Not Special in Dynamic Scoping

With dynamic scoping, recursive procedures require no special mechanism. Running the program

```
let f = proc (x) (f x)
in (f 1)
```

via dynamic scoping semantics

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

proceeds well:

$$\cfrac{\cfrac{\cfrac{\cfrac{\vdots}{[f \mapsto (x, f\ x), x \mapsto 1] \vdash \texttt{f x} \Rightarrow}}{[f \mapsto (x, f\ x), x \mapsto 1] \vdash \texttt{f x} \Rightarrow}}{[f \mapsto (x, f\ x)] \vdash \texttt{f 1} \Rightarrow}}{[] \vdash \texttt{let f = proc (x) (f x) in (f 1)} \Rightarrow}$$

Adding Recursive Procedures

$$
\begin{array}{rcl}
P & \rightarrow & E \\
E & \rightarrow & n \\
& | & x \\
& | & E + E \\
& | & E - E \\
& | & \texttt{iszero}\ E \\
& | & \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
& | & \texttt{let}\ x = E\ \texttt{in}\ E \\
& | & \texttt{read} \\
& | & \texttt{letrec}\ f(x) = E\ \texttt{in}\ E \\
& | & \texttt{proc}\ x\ E \\
& | & E\ E
\end{array}
$$

Example

```
letrec double(x) =
  if iszero(x) then 0 else ((double (x-1)) + 2)
in (double 1)
```

Semantics of Recursive Procedures

- Domain:

$$\begin{array}{rcl} Val & = & \mathbb{Z} + Bool + Procedure + RecProcedure \\ Procedure & = & Var \times E \times Env \\ RecProcedure & = & Var \times Var \times E \times Env \\ Env & = & Var \to Val \end{array}$$

- Semantics rules:

$$\frac{[f \mapsto (f, x, E_1, \rho)]\rho \vdash E_2 \Rightarrow v}{\rho \vdash \texttt{letrec}\ f(x) = E_1\ \texttt{in}\ E_2 \Rightarrow v}$$

$$\frac{\begin{array}{c}\rho \vdash E_1 \Rightarrow (f, x, E, \rho') \quad \rho \vdash E_2 \Rightarrow v \\ [x \mapsto v, f \mapsto (f, x, E, \rho')]\rho' \vdash E \Rightarrow v'\end{array}}{\rho \vdash E_1\ E_2 \Rightarrow v'}$$

Example

$$
\dfrac{[f \mapsto (f, x, f\ x, [])] \vdash f \Rightarrow (f, x, f\ x, []) \qquad \dfrac{\dfrac{\vdots}{[x \mapsto 1, f \mapsto (f, x, f\ x, [])] \vdash f\ x \Rightarrow}}{[x \mapsto 1, f \mapsto (f, x, f\ x, [])] \vdash f\ x \Rightarrow}}{\dfrac{[f \mapsto (f, x, f\ x, [])] \vdash f\ 1 \Rightarrow}{[] \vdash \texttt{letrec}\ f(x) = f\ x\ \texttt{in}\ f\ 1 \Rightarrow}}
$$

Mutually Recursive Procedures

$$
\begin{array}{rcl}
P & \rightarrow & E \\
E & \rightarrow & n \\
 & | & x \\
 & | & E + E \\
 & | & E - E \\
 & | & \texttt{iszero}\ E \\
 & | & \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
 & | & \texttt{let}\ x = E\ \texttt{in}\ E \\
 & | & \texttt{read} \\
 & | & \texttt{letrec}\ f(x) = E\ \texttt{in}\ E \\
 & | & \texttt{letrec}\ f(x_1) = E_1\ \text{and}\ g(x_2) = E_2\ \texttt{in}\ E \\
 & | & \texttt{proc}\ x\ E \\
 & | & E\ E
\end{array}
$$

Example

```
letrec
  even(x) = if iszero(x) then 1 else odd(x-1)
  odd(x)  = if iszero(x) then 0 else even(x-1)
in (odd 13)
```

Semantics of Recursive Procedures

To support mutually recursive procedures, we need to extend the domain and sematnics:

- Domain:

$$\begin{array}{rcl} \boldsymbol{Val} & = & \cdots + \boldsymbol{MRecProcedure} \\ \boldsymbol{MRecProcedure} & = & ? \end{array}$$

- Semantics rules:

$$\frac{?}{\rho \vdash \texttt{letrec}\ f(x) = E_1\ \texttt{and}\ g(y) = E_2\ \texttt{in}\ E_3 \Rightarrow ?}$$

$$\frac{?}{\rho \vdash E_1\ E_2 \Rightarrow ?}$$

Summary: The Proc Language

A programming language with expressions and procedures:

Syntax

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

Summary

Semantics

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

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

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

$$\frac{\rho \vdash E_1 \Rightarrow v_1 \quad [x \mapsto v_1]\rho \vdash E_2 \Rightarrow v}{\rho \vdash \texttt{let } x = E_1 \texttt{ in } E_2 \Rightarrow v} \qquad \frac{[f \mapsto (f, x, E_1, \rho)]\rho \vdash E_2 \Rightarrow v}{\rho \vdash \texttt{letrec } f(x) = E_1 \texttt{ in } E_2 \Rightarrow v}$$

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

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

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