COSE212: Programming Languages

Lecture 6 — Procedures

Hakjoo Oh
2015 Fall

Let: A Simple Expression Language

Syntax:

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

Let: A Simple Expression Language

Semantic domain:

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

Semantics rules:

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

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

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

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

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

Proc = Let + Procedures

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

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 in the body of a procedure whose formal parameter is `x`.
- Otherwise, the variable is *free*.
- In procedure

```
proc (y) (x+y)
```

  `x` is free and `y` is bound.

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 (f 3)
```

Two ways to determine free variables of procedures:

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

Most modern languages use static scoping.

Why Static Scoping?

- Dynamic scoping makes programs very difficult to understand.
  - In static scoping, names are resolved at compile-time.
  - In dynamic scoping, names are resolved during program execution.
- ex) What is the result of the program?

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

- In static scoping, renaming bound variables by their definitions does not change the semantics, which is unsafe in dynamic scoping.

Semantics of Procedures: Static Scoping

- Domain:

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

  The procedure value is called *closures*.

- Semantics rule:

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

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

Example

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

cf) 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 rule:

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

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

Example: Dynamic Scoping

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

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

Multiple Argument Procedures by Currying

- 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)
  ```

- This is called *Currying*, and the procedure is said to be *Curried*.

Recursive Procedures

Our language does not support recursive procedures:

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

Evaluation:

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

LETREC: A Language with Recursive Procedures

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

Example

```
letrec double(x) =
  if zero?(x) then 0 else ((double (x-1)) + 2)
in (double 6)
```

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 rule:

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

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

Example

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

cf) 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

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

proceeds well:

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

Summary

A "Turing-complete" language with expressions and procedures:

Syntax

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

Summary

Semantics

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

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

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

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

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

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

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