Announcements

1. Project deadline extended

1

Lecture 22
IREF

T. METIN SEZGIN

2

Implicit references

- EREF
  - References instantiated explicitly
  - References explicitly stored in the store
  - Expressed and denoted values include references

$$\begin{aligned} ExpVal &= Int + Bool + Proc + Ref(ExpVal) \\ DenVal &= ExpVal \end{aligned}$$

3

Implicit references

- IREF
  - References are instantiated by the interpreter
  - All denoted values are references to expressed values
  - Each binding operation introduces a location
    - Let
    - letrec
    - proc
  - Pointers to stores are saved in the environment

$$\begin{aligned} ExpVal &= Int + Bool + Proc \\ DenVal &= Ref(ExpVal) \end{aligned}$$

4

New grammar

- A set operation for assignment

Expression ::= `set` *Identifier* `=` *Expression*

```
assign-exp (var exp1)
```

5

Examples

```
let x = 0
in letrec even(dummy)
            = if zero?(x)
              then 1
              else begin
                     set x = -(x,1);
                     (odd 888)
                   end
          odd(dummy)
            = if zero?(x)
              then 0
              else begin
                     set x = -(x,1);
                     (even 888)
                   end
   in begin set x = 13; (odd -888) end
```

```
let g = let count = 0
        in proc (dummy)
             begin
               set count = -(count,-1);
               count
             end
in let a = (g 11)
   in let b = (g 11)
      in -(a,b)
```

6

Behavior specification

- **var-exp**

$$\texttt{(value-of (var-exp } var\texttt{) } \rho\ \sigma\texttt{)} = (\sigma(\rho(var)), \sigma)$$

- **assign-exp**

$$\frac{\texttt{(value-of } exp_1\ \rho\ \sigma_0\texttt{)} = (val_1, \sigma_1)}{\texttt{(value-of (assign-exp } var\ exp_1\texttt{)} \ \rho\ \sigma_0\texttt{)} = (\lceil 27 \rceil, [\rho(var) = val_1]\sigma_1)}$$

- **apply-procedure**

$$\texttt{(apply-procedure (procedure } var\ body\ \rho\texttt{) } val\ \sigma\texttt{)} = \texttt{(value-of } body\ [var = l]\rho\ [l = val]\sigma\texttt{)}$$

7

Implementation

- **var-exp**

```
(var-exp (var) (deref (apply-env env var)))
```

- **assign-exp**

```
(assign-exp (var exp1)
  (begin
    (setref!
      (apply-env env var)
      (value-of exp1 env))
    (num-val 27)))
```

- **apply-procedure**

apply-procedure : $Proc \times ExpVal \rightarrow ExpVal$

```
(define apply-procedure
  (lambda (proc1 val)
    (cases proc proc1
      (procedure (var body saved-env)
        (value-of body
          (extend-env var (newref val) saved-env))))))
```

8

Implementation

Reference instantiations

- **apply-procedure**

apply-procedure : *Proc* × *ExpVal* → *ExpVal*

```
(define apply-procedure
  (lambda (proc1 val)
    (cases proc proc1
      (procedure (var body saved-env)
        (value-of body
          (extend-env var (newref val) saved-env))))))
```

- **let**

```
(let-exp (var exp1 body)
  (let ((val1 (value-of exp1 env)))
    (value-of body
      (extend-env var (newref val1) env))))
```

- **letrec**

```
(extend-env-rec (p-names b-vars p-bodies saved-env)
  (let ((n (location search-var p-names)))
    (if n
      (newref
        (proc-val
          (procedure
            (list-ref b-vars n)
            (list-ref p-bodies n)
            env)))
      (apply-env saved-env search-var))))
```

9