Lecture 16
LETREC

T. METIN SEZGIN

PROC is ex; long live LETREC

- PROC had its limitations
  - No recursive procedures
- Define a language with recursive procedures
  - Specification
    - Syntax
    - Semantics
  - Representation
  - Implementation

Nuggets of the lecture

- Implementation requires creating representation for recursive procedures
- We need to rethink how we evaluate recursive procedures
- A more elaborate way of representing procedures in the environment is needed

LETREC

- The idea

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

- The new grammar

Expression ::= `letrec` *Identifier* `(`*Identifier*`)` `=` *Expression* `in` *Expression*

$\boxed{\texttt{letrec-exp (p-name b-var p-body letrec-body)}}$

LETREC

- Extend the environment recursively

```
(value-of
  (letrec-exp proc-name bound-var proc-body letrec-body)
  ρ)
= (value-of
    letrec-body
    (extend-env-rec proc-name bound-var proc-body ρ))
```

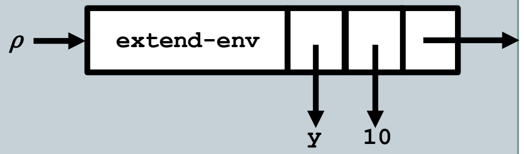
- How should environment lookup work?
  - If the search variable matches a recursive procedure

```
(apply-env ρ1 proc-name)
= (proc-val (procedure bound-var proc-body ρ1))
```

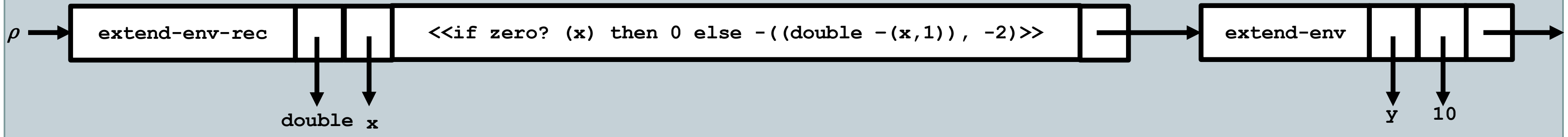
  - If there is no match

```
(apply-env ρ1 var) = (apply-env ρ var)
```

Extended environment



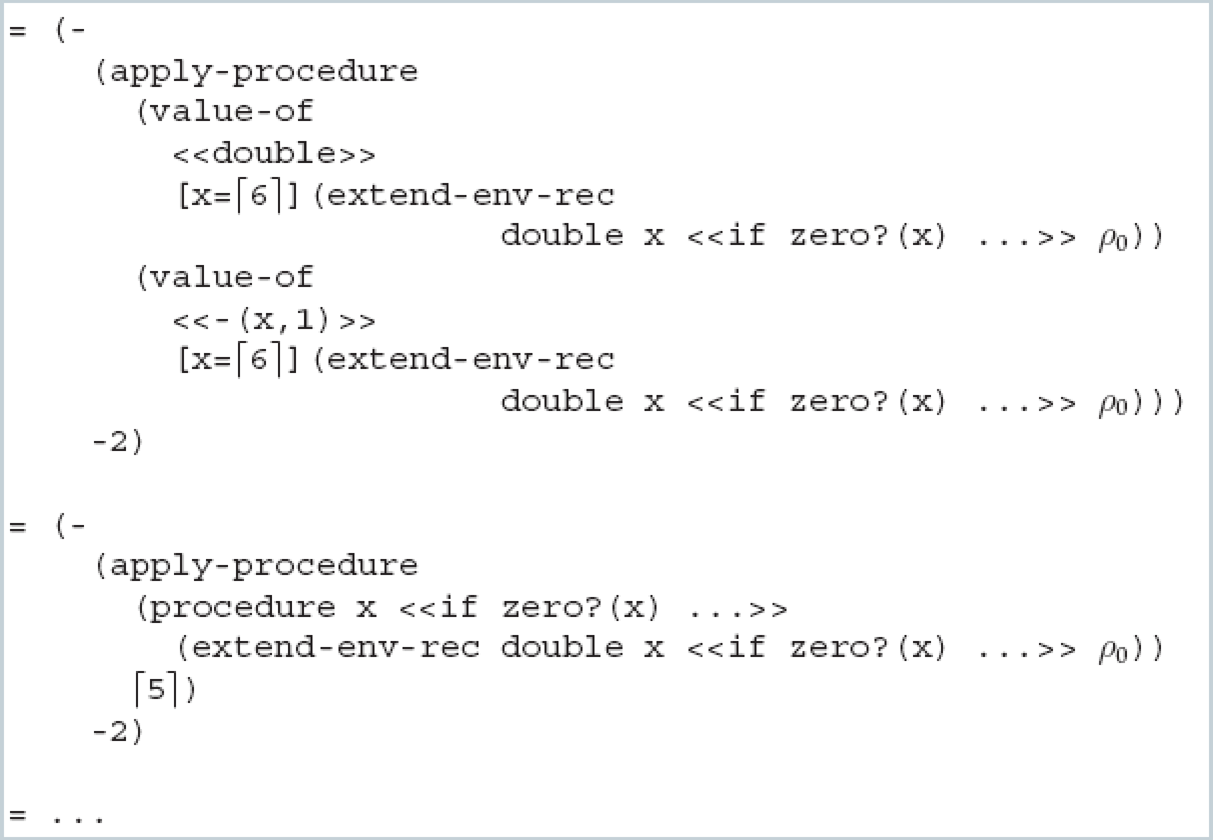
Extended environment



Example

```
(value-of <<letrec double(x) = if zero?(x)
                               then 0
                               else -((double -(x,1)), -2)
            in (double 6)>> ρ0)

= (value-of <<(double 6)>>
    (extend-env-rec double x <<if zero?(x) ...>> ρ0))

= (apply-procedure
    (value-of <<double>> (extend-env-rec double x
                           <<if zero?(x) ...>> ρ0))
    (value-of <<6>> (extend-env-rec double x
                      <<if zero?(x) ...>> ρ0)))

= (apply-procedure
    (procedure x <<if zero?(x) ...>>
      (extend-env-rec double x <<if zero?(x) ...>> ρ0))
    ⌈6⌉)

= (value-of
    <<if zero?(x) ...>>
   [x=⌈6⌉](extend-env-rec
               double x <<if zero?(x) ...>> ρ0))

...

= (-
    (value-of
      <<(double -(x,1))>>
      [x=⌈6⌉](extend-env-rec
                  double x <<if zero?(x) ...>> ρ0))
    -2)
```

Example cont.



```
= (-
    (apply-procedure
      (value-of
        <<double>>
        [x=⌈6⌉] (extend-env-rec
                        double x <<if zero?(x) ...>> $\rho_0$))
      (value-of
        <<-(x,1)>>
        [x=⌈6⌉] (extend-env-rec
                        double x <<if zero?(x) ...>> $\rho_0$)))
    -2)

= (-
    (apply-procedure
      (procedure x <<if zero?(x) ...>>
        (extend-env-rec double x <<if zero?(x) ...>> $\rho_0$))
      ⌈5⌉)
    -2)

= ...
```

```
let a=1 in
  let b=2 in
    letrec f(x) = if zero?(x) then 0 else -((f -(x,1)),-2) in (f 2)")
```

The new `environment` and `apply-env`

```
(define-datatype environment environment?
  (empty-env)
  (extend-env
    (var identifier?)
    (val expval?)
    (env environment?))
  (extend-env-rec
    (p-name identifier?)
    (b-var identifier?)
    (body expression?)
    (env environment?)))

(define apply-env
  (lambda (env search-var)
    (cases environment env
      (empty-env ()
        (report-no-binding-found search-var))
      (extend-env (saved-var saved-val saved-env)
        (if (eqv? saved-var search-var)
          saved-val
          (apply-env saved-env search-var)))
      (extend-env-rec (p-name b-var p-body saved-env)
        (if (eqv? search-var p-name)
          (proc-val (procedure b-var p-body env))
          (apply-env saved-env search-var))))))
```