Announcements

1. No physical lectures next week – online only

Lecture 15
PROC – Implementation

T. Metin Sezgin

Nuggets of the lecture

- Implementation requires creating representation for procs
- We can use scheme representation (procedural)
- We can use data-structure-based representation
- We need to extend the values and value-of

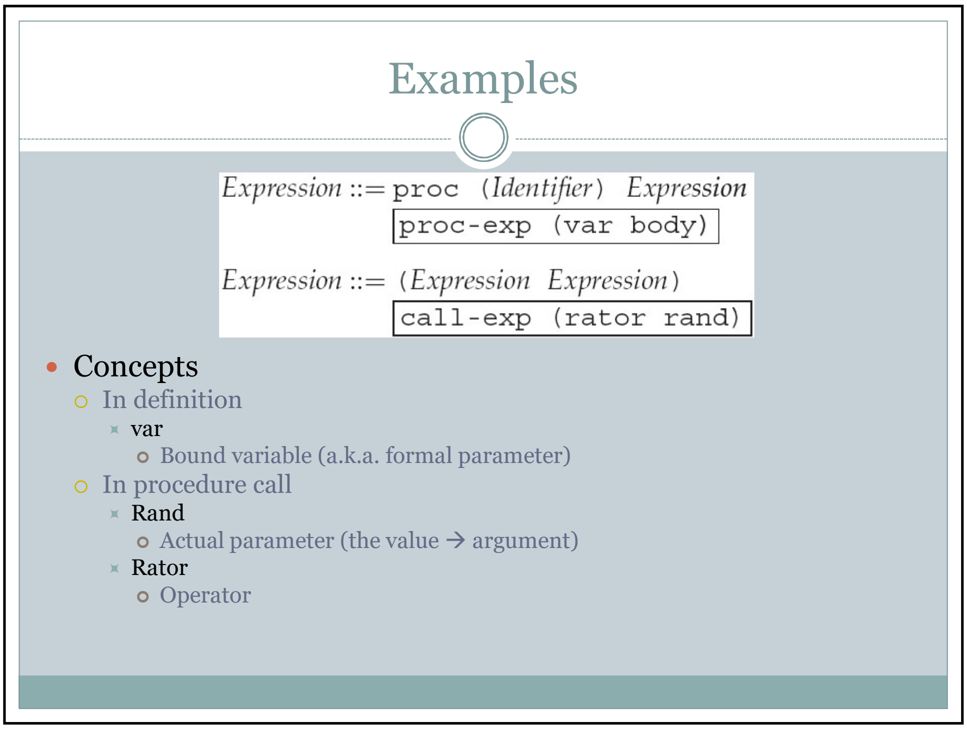
Expressed and Denoted values

- Before

$$\begin{aligned} ExpVal &= Int + Bool \\ DenVal &= Int + Bool \end{aligned}$$

- After

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



Examples

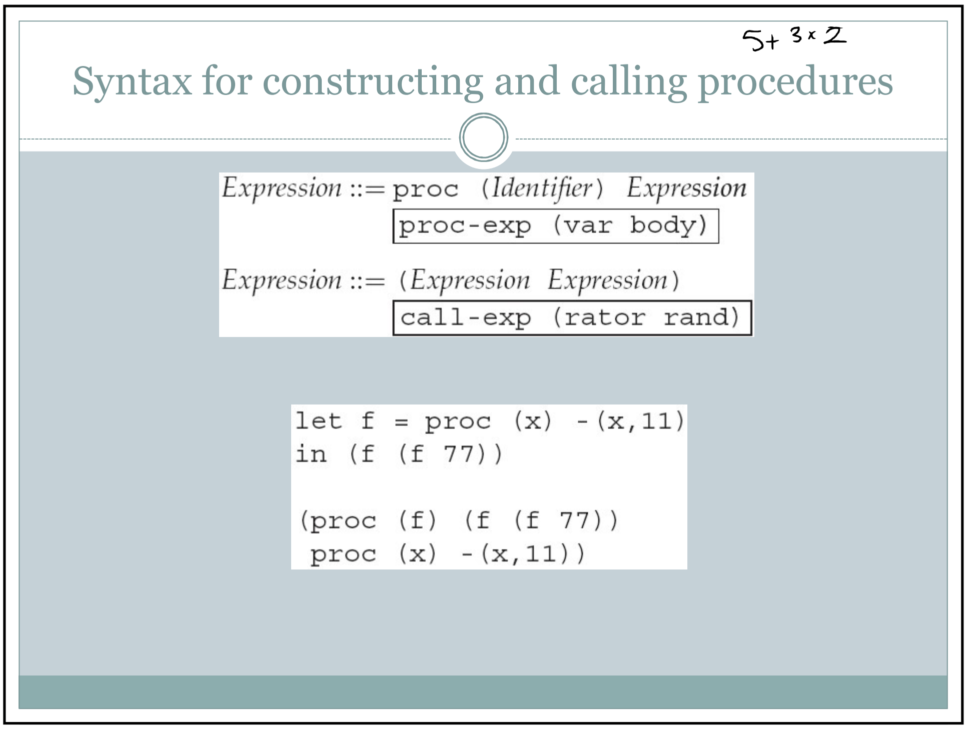
Expression ::= `proc (`*Identifier*`)` *Expression*

`proc-exp (var body)`

Expression ::= `(`*Expression* *Expression*`)`

`call-exp (rator rand)`

- Concepts
  - In definition
    - var
      - Bound variable (a.k.a. formal parameter)
  - In procedure call
    - Rand
      - Actual parameter (the value → argument)
    - Rator
      - Operator



5+ 3 × 2

Syntax for constructing and calling procedures

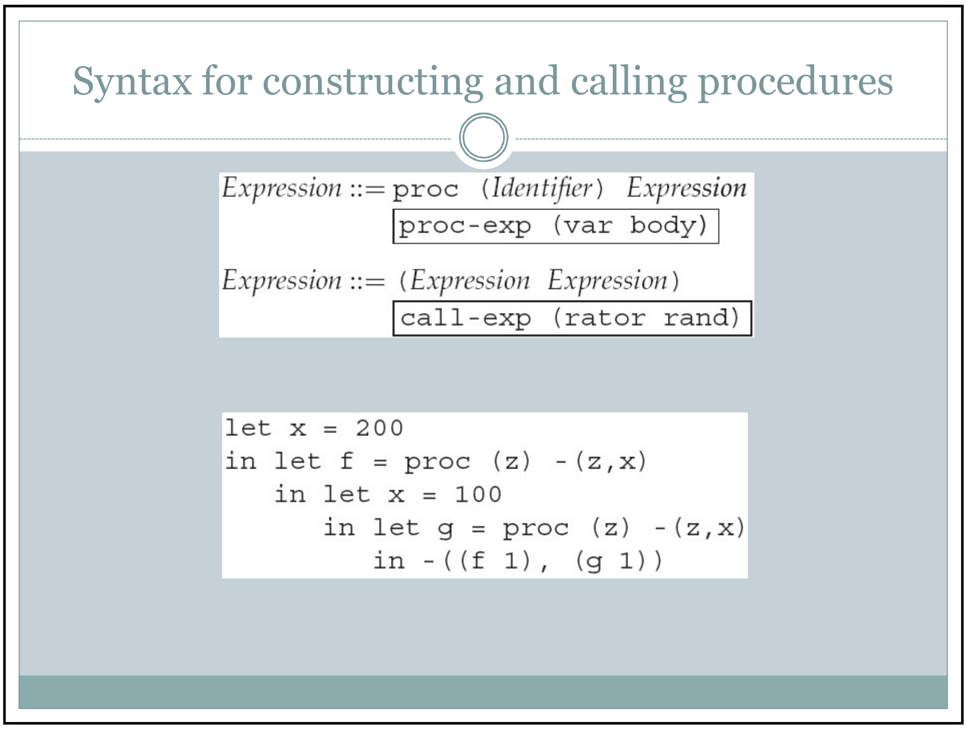
Expression ::= `proc (`*Identifier*`)` *Expression*

`proc-exp (var body)`

Expression ::= `(`*Expression* *Expression*`)`

`call-exp (rator rand)`

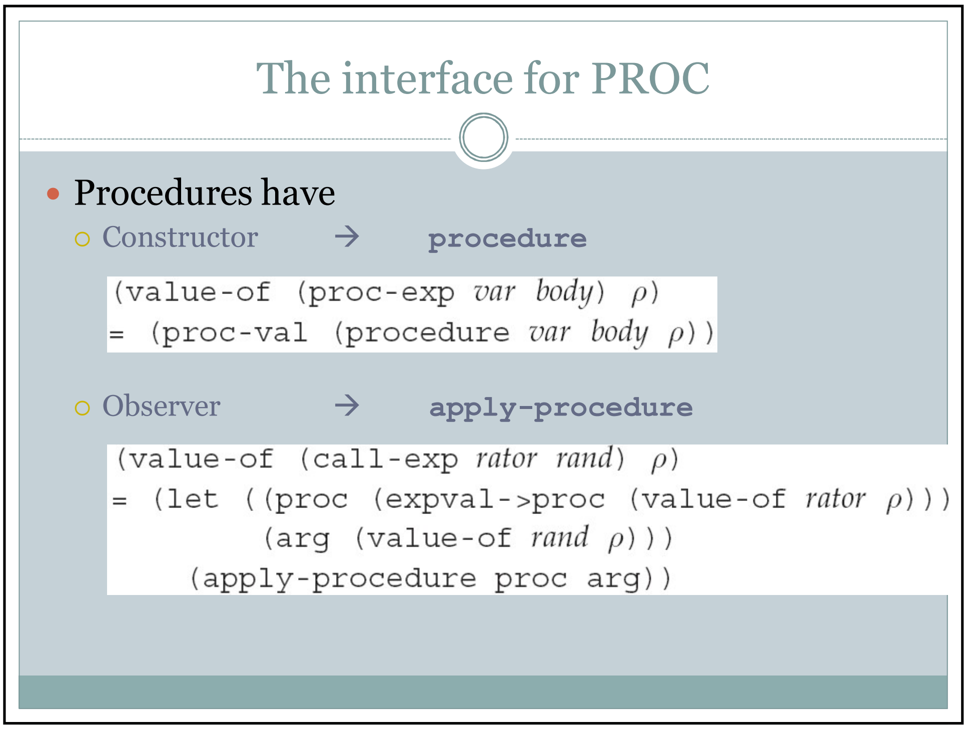
```
let f = proc (x) -(x,11)
in (f (f 77))

(proc (f) (f (f 77))
 proc (x) -(x,11))
```



Syntax for constructing and calling procedures

```
Expression ::= proc (Identifier) Expression
               proc-exp (var body)

Expression ::= (Expression Expression)
               call-exp (rator rand)
```

```
let x = 200
in let f = proc (z) -(z,x)
   in let x = 100
      in let g = proc (z) -(z,x)
         in -((f 1), (g 1))
```



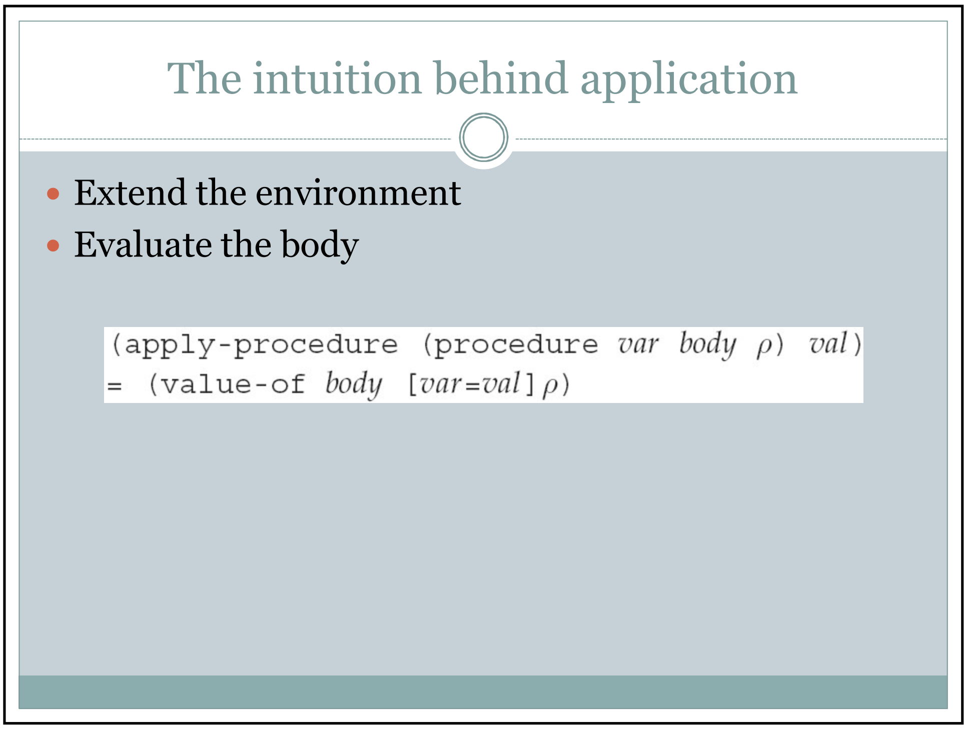
The interface for PROC

- Procedures have
  - Constructor → **procedure**

```
(value-of (proc-exp var body) ρ)
= (proc-val (procedure var body ρ))
```

  - Observer → **apply-procedure**

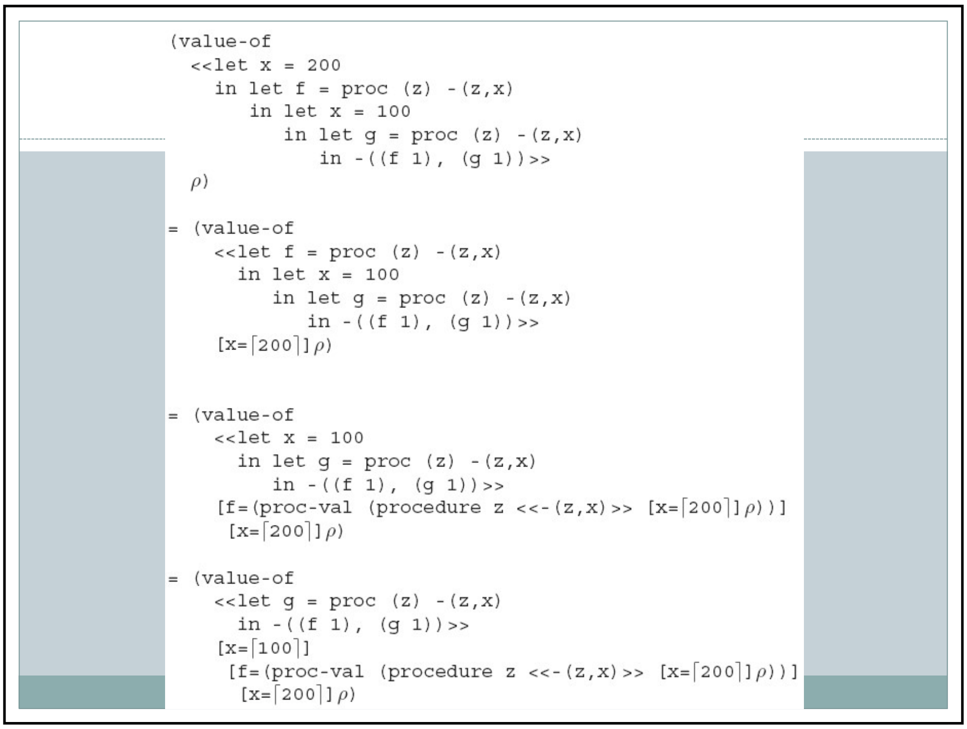
```
(value-of (call-exp rator rand) ρ)
= (let ((proc (expval->proc (value-of rator ρ)))
        (arg (value-of rand ρ)))
    (apply-procedure proc arg))
```



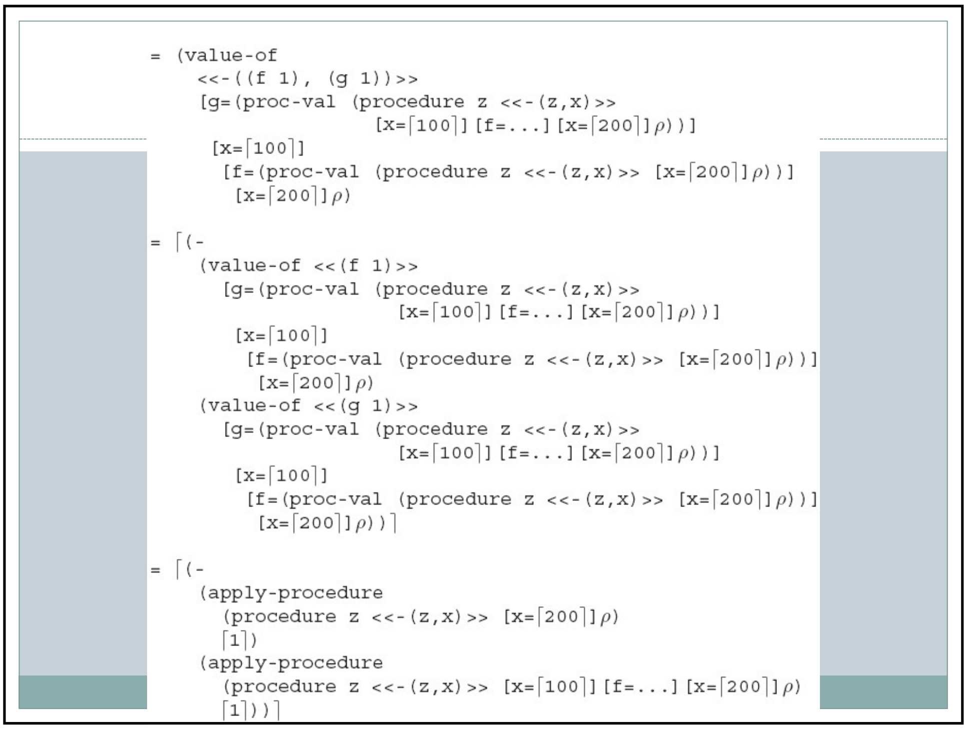
The intuition behind application

- Extend the environment
- Evaluate the body

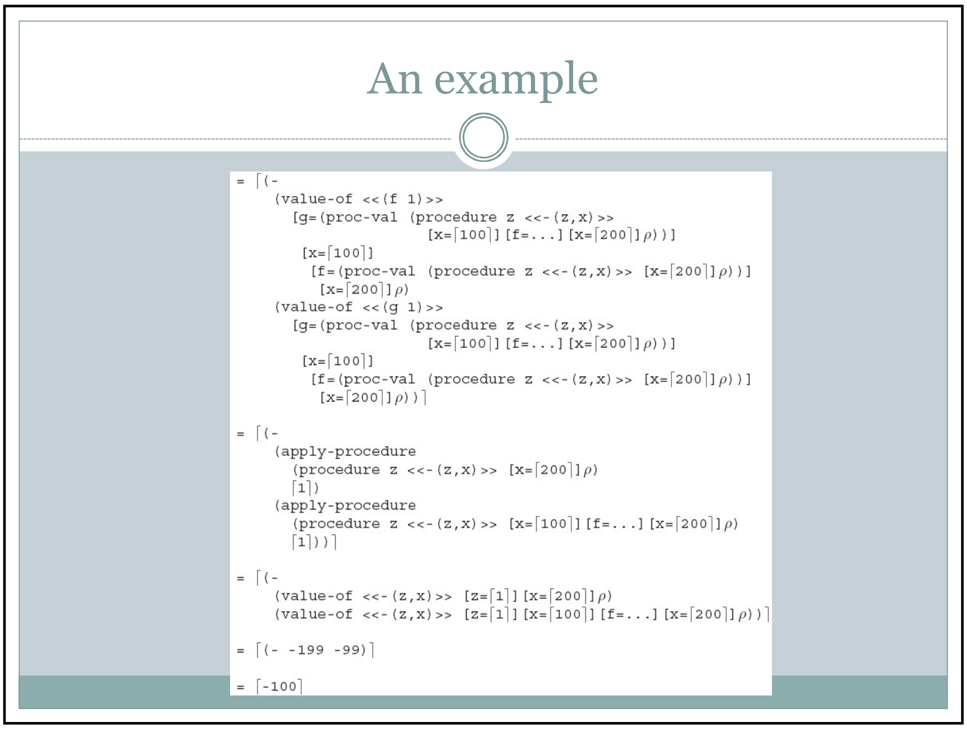
```
(apply-procedure (procedure var body ρ) val)
= (value-of body [var=val]ρ)
```



```
(value-of
  <<let x = 200
    in let f = proc (z) -(z,x)
       in let x = 100
          in let g = proc (z) -(z,x)
             in -((f 1), (g 1))>>
  ρ)

= (value-of
    <<let f = proc (z) -(z,x)
      in let x = 100
         in let g = proc (z) -(z,x)
            in -((f 1), (g 1))>>
    [x=⌈200⌉]ρ)


= (value-of
    <<let x = 100
      in let g = proc (z) -(z,x)
         in -((f 1), (g 1))>>
    [f=(proc-val (procedure z <<-(z,x)>> [x=⌈200⌉]ρ))]
     [x=⌈200⌉]ρ)

= (value-of
    <<let g = proc (z) -(z,x)
      in -((f 1), (g 1))>>
    [x=⌈100⌉]
     [f=(proc-val (procedure z <<-(z,x)>> [x=⌈200⌉]ρ))]
      [x=⌈200⌉]ρ)
```

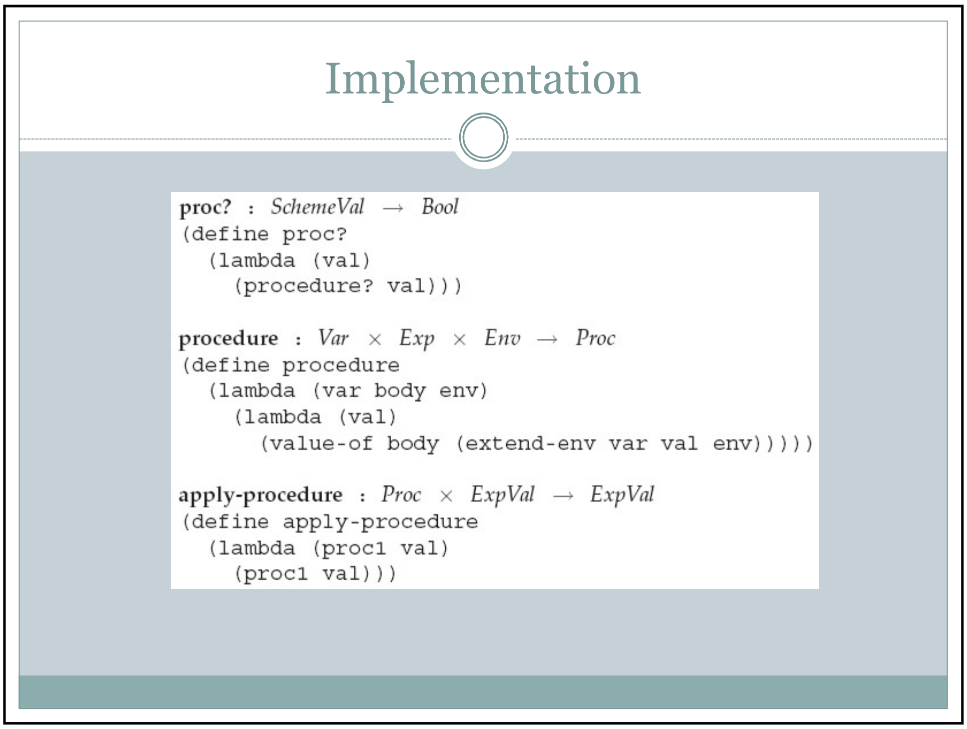


```
= (value-of
    <<-((f 1), (g 1))>>
    [g=(proc-val (procedure z <<-(z,x)>>
                   [x=⌈100⌉][f=...][x=⌈200⌉]ρ))]
     [x=⌈100⌉]
      [f=(proc-val (procedure z <<-(z,x)>> [x=⌈200⌉]ρ))]
       [x=⌈200⌉]ρ)

= ⌈(-
    (value-of <<(f 1)>>
      [g=(proc-val (procedure z <<-(z,x)>>
                     [x=⌈100⌉][f=...][x=⌈200⌉]ρ))]
       [x=⌈100⌉]
        [f=(proc-val (procedure z <<-(z,x)>> [x=⌈200⌉]ρ))]
         [x=⌈200⌉]ρ)
    (value-of <<(g 1)>>
      [g=(proc-val (procedure z <<-(z,x)>>
                     [x=⌈100⌉][f=...][x=⌈200⌉]ρ))]
       [x=⌈100⌉]
        [f=(proc-val (procedure z <<-(z,x)>> [x=⌈200⌉]ρ))]
         [x=⌈200⌉]ρ))⌉

= ⌈(-
    (apply-procedure
      (procedure z <<-(z,x)>> [x=⌈200⌉]ρ)
      ⌈1⌉)
    (apply-procedure
      (procedure z <<-(z,x)>> [x=⌈100⌉][f=...][x=⌈200⌉]ρ)
      ⌈1⌉))⌉
```



An example

```
= ⌈(-
    (value-of <<(f 1)>>
      [g=(proc-val (procedure z <<-(z,x)>>
                     [x=⌈100⌉][f=...][x=⌈200⌉]ρ))]
       [x=⌈100⌉]
        [f=(proc-val (procedure z <<-(z,x)>> [x=⌈200⌉]ρ))]
         [x=⌈200⌉]ρ)
    (value-of <<(g 1)>>
      [g=(proc-val (procedure z <<-(z,x)>>
                     [x=⌈100⌉][f=...][x=⌈200⌉]ρ))]
       [x=⌈100⌉]
        [f=(proc-val (procedure z <<-(z,x)>> [x=⌈200⌉]ρ))]
         [x=⌈200⌉]ρ))⌉

= ⌈(-
    (apply-procedure
      (procedure z <<-(z,x)>> [x=⌈200⌉]ρ)
      ⌈1⌉)
    (apply-procedure
      (procedure z <<-(z,x)>> [x=⌈100⌉][f=...][x=⌈200⌉]ρ)
      ⌈1⌉))⌉

= ⌈(-
    (value-of <<-(z,x)>> [z=⌈1⌉][x=⌈200⌉]ρ)
    (value-of <<-(z,x)>> [z=⌈1⌉][x=⌈100⌉][f=...][x=⌈200⌉]ρ))⌉

= ⌈(- -199 -99)⌉

= ⌈-100⌉
```



Implementation

proc? : *SchemeVal* → *Bool*

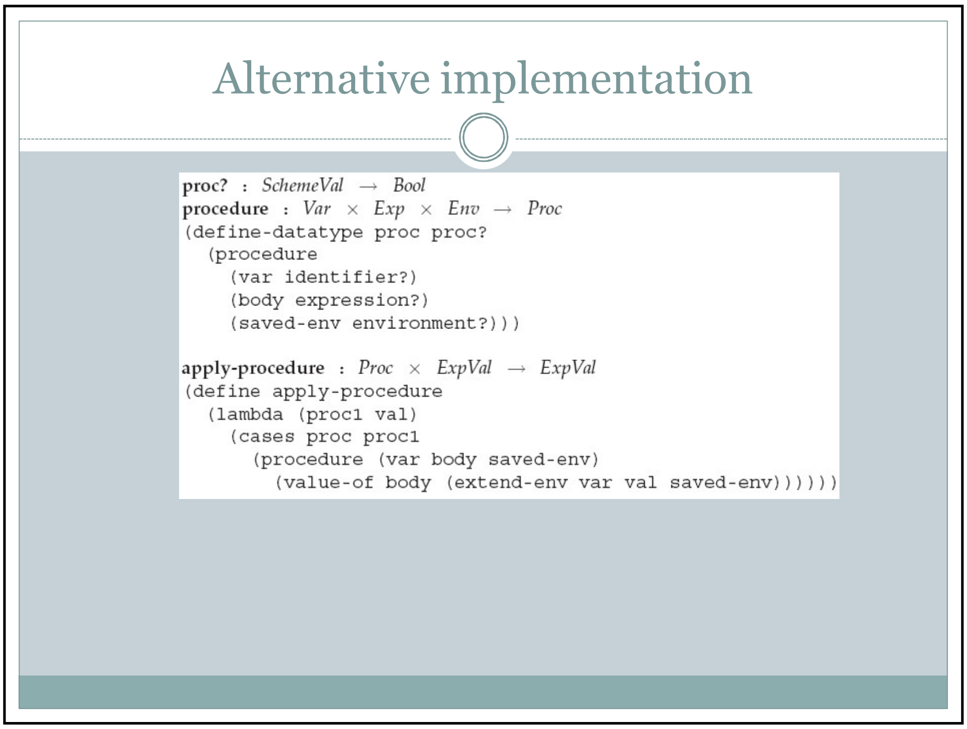
```
(define proc?
  (lambda (val)
    (procedure? val)))
```

procedure : *Var* × *Exp* × *Env* → *Proc*

```
(define procedure
  (lambda (var body env)
    (lambda (val)
      (value-of body (extend-env var val env)))))
```

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

```
(define apply-procedure
  (lambda (proc1 val)
    (proc1 val)))
```



Alternative implementation

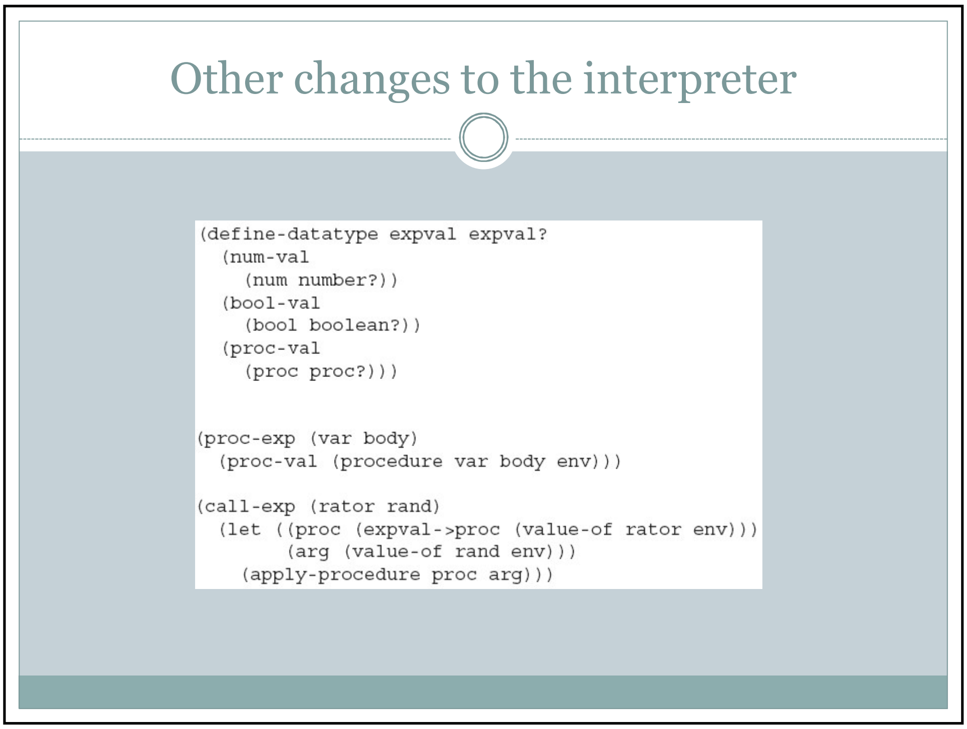
proc? : *SchemeVal* → *Bool*

procedure : *Var* × *Exp* × *Env* → *Proc*

```
(define-datatype proc proc?
  (procedure
    (var identifier?)
    (body expression?)
    (saved-env environment?)))
```

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 val saved-env))))))
```



Other changes to the interpreter

```
(define-datatype expval expval?
  (num-val
    (num number?))
  (bool-val
    (bool boolean?))
  (proc-val
    (proc proc?)))


(proc-exp (var body)
  (proc-val (procedure var body env)))

(call-exp (rator rand)
  (let ((proc (expval->proc (value-of rator env)))
        (arg (value-of rand env)))
    (apply-procedure proc arg)))
```