Adding Procedures to your Language

Principles of Programming Languages

Doug Blank

Nov 6, 2008

Concrete Syntax

```
define x = 1
```

Abstract Syntax

define x = 1

```
(declaration-statement
    (id x)
    (lit-exp 1))
```

Concrete Syntax

proc (n) 1

Abstract Syntax

proc (n) 1

```
(proc-exp
    ((id n))
    (lit-exp 1))
```

Concrete Syntax

proc (n) 1

Concrete Syntax

```
define function =
    proc (n) +(n, 1)
```

Abstract Syntax

```
define function =
    proc (n) +(n, 1)

(declaration-statement
    (id function)
    (proc-exp
        ((id n))
        (app-exp
            (var-exp +)
            ((var-exp n) (lit-exp 1)))))
```

Interpreting proc

“proc (n) n”

```
(proc-exp
    ((id n))
    (app-exp
        (var-exp +)
        ((var-exp n))))
```

What happens when you evaluate this?

Interpreting proc

> (lambda (n) n)

Interpreting proc

> (lambda (n) n)
#<procedure>

What is a procedure?

How can we use proc?

```
--> define incrementer =
        proc (n)
            proc (x)
                +(x, n)
ok
--> define add7 = incrementer(7)
ok
--> add7(100)
107
```

How can we use proc?

```
--> define add1000 = incrementer(1000)
ok
--> add1000(234)
1234
```

A proc evaluates to a *closure*

concrete:
   “proc (n) n”

parses gives:
   (proc-exp ((id n)) (var-exp n))

m gives:
   (closure-exp ((id n)) (var-exp n) *env*)

where *env* is the *env* at time of interpreting

Applying a closure

```
---> define identity = proc (n) n
ok
---> identity(8)
8
```

```
(apply-exp
   (closure-exp ((id n)) (var-exp n) env)
   ((lit-exp 8)))
```

Testing a closure

```
--> define incrementer =
        proc (n)
            proc (x)
                +(x, n)
ok
--> define add7 = incrementer(7)
ok
--> add7(100)
107
```