Scheme Notes 01

Geoffrey Matthews

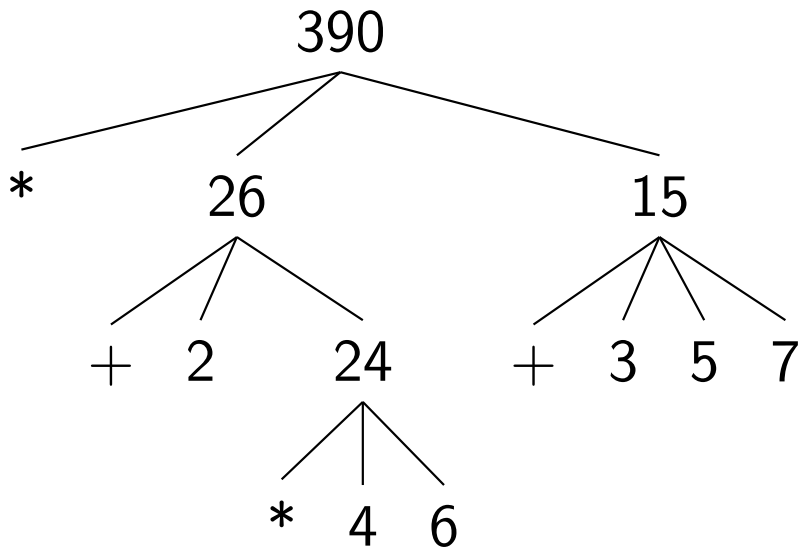
Department of Computer Science
Western Washington University

January 5, 2017

Program Evaluation In Lisp

A process of *tree accumulation.*

```
(* (+ 2 (* 4 6))
   (+ 3 5 7))
```



Introducing Local Variables

```
(let ((x 3)
      (y 4)
      (z 5))
    (+ x (* y z)))   =>  23
```

Beware! This will NOT work.

```
(let ((x 3)
      (y (* 2 x))
      (z (* 3 x)))
    (+ x (* y z)))   =>  57
```

But this will.

```
(let* ((x 3)
       (y (* 2 x))
       (z (* 3 x)))
    (+ x (* y z)))   =>  57
```

Defining Procedures

Two equivalent ways:

```
(define (square x) (* x x))
(define square (lambda (x) (* x x)))
```

The first one is more in line with the procedure call:

```
(square 5) => 25
```

Defining Procedures

Two equivalent ways:

```
(define (square x) (* x x))
(define square (lambda (x) (* x x)))
```

The second one is more in line with defining other things:

```
(define x (* 3 4))
(define y (list 'a 'b 'c))
```

The action of `define` is simply to give a *name* to the result of an expression.

Defining Procedures

Two equivalent ways:

```
(define (square x) (* x x))
(define square (lambda (x) (* x x)))
```

The result of a lambda-expression is an anonymous function. We can name it, as above, or use it without any name at all:

```
(square 5) => 25
((lambda (x) (* x x)) 5) => 25
```

Solving problems

Newton's method:
If y is a guess for $\sqrt{x}$, then the average of y and x/y is an even better guess.

x	guess	quotient	average
2	1.0	2.0	1.5
2	1.5	1.3333333333333333	1.4166666666666665
2	1.4166666666666665	1.411764705882353	1.4142156862745097
2	1.4142156862745097	1.41421143847487	1.414213562374689 9

...

Evidently, we want to iterate, and keep recomputing these things until we find a value that's close enough.

Newton's Method in Scheme

```
(define sqrt-iter
  (lambda (guess x)
    (if (good-enough? guess x)
        guess
        (sqrt-iter (improve guess x) x))))

(define improve
  (lambda (guess x)
    (average guess (/ x guess))))

(define average
  (lambda (x y) (/ (+ x y) 2)))

(define good-enough?
  (lambda (guess x)
    (< (abs (- (square guess) x)) 0.00001)))

(define square
  (lambda (x) (* x x)))

(define sqrt
  (lambda (x) (sqrt-iter 1.0 x)))
```

Decompose big problems into smaller problems.

Definitions can be nested

```
(define sqrt
  (lambda (x)
    (define good-enough?
      (lambda (guess x)
        (< (abs (- (square guess) x)) 0.001)))
    (define improve
      (lambda (guess x)
        (average guess (/ x guess))))
    (define sqrt-iter
      (lambda (guess x)
        (if (good-enough? guess x)
            guess
            (sqrt-iter (improve guess x) x))))
    (sqrt-iter 1.0 x)))
```

Parameters need not be repeated

```
(define sqrt
  (lambda (x)
    (define good-enough?
      (lambda (guess)
        (< (abs (- (square guess) x)) 0.001)))
    (define improve
      (lambda (guess)
        (average guess (/ x guess))))
    (define sqrt-iter
      (lambda (guess)
        (if (good-enough? guess)
            guess
            (sqrt-iter (improve guess)))))
    (sqrt-iter 1.0)))
```

Introducing local functions with `letrec`

```
(define sqrt
  (lambda (x)
    (letrec ((good-enough?
              (lambda (guess)
                (< (abs (- (square guess) x)) 0.001)))
             (improve
              (lambda (guess)
                (average guess (/ x guess))))
             (sqrt-iter
              (lambda (guess)
                (if (good-enough? guess)
                    guess
                    (sqrt-iter (improve guess))))))
             )
             (sqrt-iter 1.0))))
```

Procedures as parameters

Summation notation:

$$\sum_{i=a}^{b} f(i) = f(a) + \ldots + f(b)$$

In scheme:

```
(define sum
  (lambda (a b f)
    (if (> a b)
        0
        (+ (f a) (sum (+ a 1) b f)))))

(sum 1 10 square) => 385
(sum 1 10 (lambda (x) (* x x x))) => 3025
```

Finding fixed points

x is a *fixed point* of f if $x = f(x)$
For some functions you can find fixed points by iterating:
$x, f(x), f(f(x)), f(f(f(x))), \ldots$

Fixed points in scheme:

```
(define fixed-point
  (lambda (f)
    (let
        ((tolerance 0.0001)
         (max-iterations 10000))
      (letrec
          ((close-enough?
            (lambda (a b) (< (abs (- a b)) tolerance)))
           (try
            (lambda (guess iterations)
              (let ((next (f guess)))
                (cond ((close-enough? guess next) next)
                      ((> iterations max-iterations) #f)
                      (else (try next (+ iterations 1)))))))))
           )
        (try 1.0 0)))))

(fixed-point cos)  => 0.7390547907469174
(fixed-point sin)  => 0.08420937654137994
(fixed-point (lambda (x) x)) => 1.0
(fixed-point (lambda (x) (+ x 1))) => #f
```

Remember Newton's Method?

```
(define sqrt
  (lambda (x)
    (fixed-point (lambda (y) (/ (+ y (/ x y)) 2)))))
```

Procedures as Returned Values

```
(define average-damp
  (lambda (f)
    (lambda (x) (/ (+ x (f x)) 2))))

(define sqrt
  (lambda (x)
    (fixed-point (average-damp (lambda (y) (/ x y))))))
```