MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring Semester, 1997

Lecture Notes – Feb. 13, 1997

Capturing Common Patterns with Higher Order Procedures

A simple common pattern: $(* \ 2 \ 2)$ $(* \ 3 \ 3)$ $(* \ 4 \ 4)$

This is captured by

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

This provides a **NAME** for the idea of multiplying something by itself.

Three sums

$$1 + 2 + \ldots + 100$$

$$1^2 + 2^2 + \ldots + 100^2$$

$$\frac{1}{1^2} + \frac{1}{3^2} + \ldots + \frac{1}{99^2}$$

Here are three procedures that compute these sums:

```
(define (sum-integers a b)
  (if (> a b)
      0
      (+ a (sum-integers (1+ a) b))))

(define (sum-squares a b)
  (if (> a b)
      0
      (+ (square a)
         (sum-squares (1+ a) b))))

(define (pi-sum a b)
  (if (> a b)
      0
      (+ (/ 1 (square a))
         (pi-sum (+ a 2) b))))
```

Notice that the type of each procedure is:

$$(\text{Sch-Num} \times \text{Sch-Num}) \mapsto \text{Sch-Num}$$

This pattern is capture by the following higher-order procedure (fill in the blank as we go along):

```
(define sum
```

Notice the unusual time of this procedure:

$$\text{F} \times \text{Sch-Num} \times \text{F} \times \text{Sch-Num} \mapsto \text{Sch-Num}$$

where F = Sch-Num $\mapsto$ Sch-num.

Use `sum` to express the three procedures above as instances of the general idea of summing:

```
(define sum-integers1
```

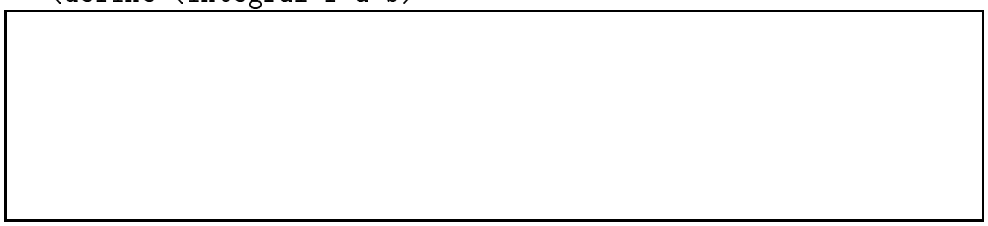
```
(define sum-squares1
```

```
(define pi-sum1
```

Now use sum to perform numerical integration:

$$\int_a^b f = [f(a) + f(a + dx) + \cdots + f(b)]\,dx$$

```
(define (integral f a b)
```



Computing square roots (again)

The square root of x is a fixed point of the transformation:

Method for finding a fixed point of a function f (that is, a value of y such that f(y)=y)

- start with a guess for y
- keep applying f over and over until the result doesn't change very much

```
(define tolerance 0.0001)

(define (close? u v)
  (< (abs (- u v)) tolerance))

(define (fixed-point f i-guess)
  (define (try g)
     (let ((next-guess (f g)))
       (if (close? next-guess g)
           next-guess
           (try next-guess))))
  (try i-guess))
```

We would like to compute the square root of x as a fixed point of the function $y \rightarrow \frac{x}{y}$.

```
(define (sqrt x)
  (fixed-point
     (lambda (y) (/ x y))
     1))
```

but this doesn't work, because it doesn't converge – it oscillates.

One way to control oscillations is to use average damping:

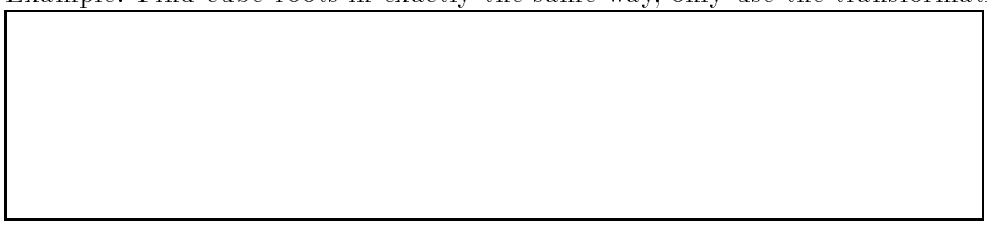
```
(define (average-damp f)
  (lambda (x)
     (average x (f x))))
```

Now we can define `sqrt` as a fixed point, and the sequence will converge:

```
(define (sqrt x)
  (fixed-point
    (average-damp
       (lambda (y) (/ x y)))
    1))
```

This is the same PROCESS as the square root example from last time, but we have EXPRESSED it differently.

The advantage of this is that we see square root more clearly as just one instance of a general idea. Example: Find cube roots in exactly the same way, only use the transformation:



Newton's method is another general method:

To find a zero of a function f (that is, a value y such that f(y) = 0), find a fixed-point of the function:

$$y \mapsto y - \frac{f(y)}{Df(y)}$$

We can compute the square root of x by using Newton's Method to find a zero of the function $y \to y^2 - x$.

The following procedure takes a procedure f as argument, and returns the derivative of f, which is itself a procedure:

```
(define deriv
  (lambda (f)
     (let ((dx 0.00001))
        (lambda (x)
          (/ (- (f (+ x dx))
                (f x))
             dx)))))
```

Example:

```
((deriv square) 10) ---> 20.00000999942131
```

Now that we have derivatives, we can express the function $x \to x - \frac{f(x)}{Df(x)}$ used in Newton's method:

$$x \to \frac{f(x+dx) - f(x)}{dx} \qquad x \to x - \frac{f(x)}{Df(x)}$$

```
(define (newton-transform f)
  (lambda (y)
     (- y
        (/ (f y)
           ((deriv f) y)))))
```

Expressin Newton's Method as finding a fixed point of the transformed function:

```
(define (newtons-method f guess)
  (fixed-point
    (newton-transform f)
    guess))
```

Now we have two different ways of expressing the square root computation:

As a fixed point process:

```
(define (sqrt x)
  (fixed-point
     (average-damp
       (lambda (y) (/ x y)))
     1))
```

As an application of Newton's method:

```
(define (sqrt x)
  (newtons-method
    (lambda (y) (- (square y) x))
    1))
```

An even more general idea is to find the fixed point of a transformed function:

```
(define (fixed-point-of-transform f transform guess)
  (fixed-points (transform f) guess))
```

This idea is general enough to express both methods for computing square roots:

```
(define (sqrt x)
  (fixed-point-of-transform
     (lambda (y) (- (square y) x))
     newton-transform
     1))

(define (sqrt x)
  (fixed-point-of-transform
     (lambda (y) (/ x y))
     average-damp
     1))
```

Rights and privileges of first class citizens (Christopher Strachey 1916–1975):

- May be named by variables
- May be passed as arguments to procedures
- May be returned as the results of procedures
- May be included in data structures