Week 2 Lecture 6

Theory

Getting Ready

- Feel good about Lecture 5
- Read SICP Section 2.2 closely

What’s in this lecture?

- Advanced list processing & structural recursion in Scheme

List Processing

- Writing functions to process lists involves a lot of boilerplate code
- Much of this boilerplate code can be eliminated by using abstractions
- In this lecture, we use introduce *map*, *foldl* & *foldr*

Map

- *map* turns a list into another list, element by element
- How does it do this? It uses a function that takes a single element as input and returns another item as output
- To turn ‘(1 2 3) into ‘(1 4 9), we would use the *square* function

Map Functions

- Define the *square* function
- Define the *halve* function
- Define the *identity* function
- Define a *number-to-string* function
- Define a *string-to-number* function

Definition of Map

Map takes a list of elements x and returns a list of elements f(x):

```
(define (map f alist) ...)
```

What do these return?

```
(map *square* '(1 2 3))
(map *halve* '(2 4 6))
```

Implementation of Map

```
;; called ‘mymap’ so it doesn’t conflict with built-in

(define (mymap f alist)
  (define (map-iter f accum alist)
    (if (null? alist)
      accum
      (cons (f (car alist)) (map-iter f accum (cdr alist)))))
  (map-iter f () alist))
```

Introducing Foldl

- When we build *map*, we used an internal function map-iter
- As it turns out, we can modify this function slightly to be very useful in general
- The generalized version is called *foldl*

Using Foldl

```
;; returns 6
(foldl + 0 ‘(1 2 3))

;; returns 48
(foldl * 1 ‘(2 4 6))

;; returns 4
(foldl (lambda (x a) (+ a 1)) 0 ‘(1 2 3 4))
```

Implementing Foldl

```
;; *foldl* takes function f(x, accum) returning new accum,
;; an initial accum, and a list

(define (foldl f accum alist)
  (if (null? alist)
    accum
    (foldl f (f (car alist) accum) (cdr alist))))
```

Length of a List

How do we define the length of a list using foldl?

```
(define (length alist)
  (foldl ...))
```

Contains using foldl

- Can we implement the *contains* function using foldl? Is it more or less efficient than a custom version?

Exercises

- Read SICP 2.2 closely (again)
- Implement fold-right, which processes the elements of the list in *reverse* order
- SICP 2.24, 2.25, 2.26, 2.27