Chapter 10 :: Functional Languages

Higher Order Functions and Conclusions

Michael L. Scott

ELSEVIER

High-Order Functions

- Higher-order functions
  - Take a function as argument, or return a function as a result
  - Great for building things

Building Things in C

- sort and search take a comparison function

```
int compare_int(void *a, void *b) {
    int x = *(int *)a;
    int y = *(int *)b;
    return x – y;
    }
int temperatures[20];
qsort(temperatures, 20, compare_int)
```

Map Function

- Takes a function and a sequence of lists, applies function pair-wise to each element of the lists, and returns a list as the result
- Example:

  (map * ‘(2 4 6) ‘(3 5 7)) ➔ (6 20 42)

Reduce (fold) Function

- Reduce a list of values to a single value using a binary operator
- Example:

```
(define fold
    (lambda (fct identity-value sequence)
        (if (null? sequence)
            identity-value ; e.g., 0 for +, 1 for *
            (fct (car sequence)
                 (fold fct identity-value (cdr sequence))))))
```

(fold * 1 '(2 4 6)) ==> 48

Using map/fold in tandem

- Matrix Multiplication

5	2	4	6	10
1	2	10	12	17
4	8	3	8	20
11	15	9	2	1

*

3	17	22
6	5	4
2	3	2
6	11	7
4	8	9

(fold + 0 (map * row column))

(+ (1*17, 2*5, 10*3, 12*11, 17*8)
➔ (+ (17, 10, 30, 132, 136))
➔ 325

Currying

- Replaces one of a function's arguments with a constant value and returns a function that accepts one fewer arguments
  - Good for creating simpler looking functions
- Simple Example

```
(define curried-plus (lambda (a) (lambda (b) (+ a b))))
   ((curried-plus 3) 4) ==> ((lambda (b) (+ 3 b)) 4) ==> 7
```

- Syntactic Sugar Example

```
(define total (lambda (L) (fold  + 0 L))
(total '( 1 2 3 4 5)) ➔ 15
```

Functional Programming in Perspective

- Advantages of functional languages
  - lack of side effects makes programs easier to understand
  - lack of explicit evaluation order (in some languages) offers possibility of parallel evaluation (e.g. MultiLisp)
  - lack of side effects and explicit evaluation order simplifies some things for a compiler (provided you don't blow it in other ways)
  - programs are often surprisingly short
  - language can be extremely small and yet powerful

Functional Programming in Perspective

- Problems
  - Performance
    - trivial update problem
      - initialization of complex structures
      - summarization problem
      - in-place mutation
    - heavy use of pointers (locality problem)
    - frequent procedure calls
    - heavy space use for recursion
    - requires garbage collection
  - requires a different mode of thinking by the programmer
  - difficult to integrate I/O into purely functional model