
A rigorous discussion of functional programming in general can be found in Henderson (1980). The process of implementing functional languages through graph reduction is discussed in detail in Peyton Jones (1987).

REVIEW QUESTIONS

1. Define functional form and referential transparency.
2. What data types were part of the original LISP?
3. What is the difference between EQ?, EQV?, and =?
4. What are the differences between the evaluation method used for the Scheme special form, DEFINE, and that used for its primitive functions?
5. What are the two forms of DEFINE?
6. Describe the semantics of COND.
7. Describe the semantics of LET.
8. Why were imperative features added to most dialects of LISP?
9. In what ways are COMMON LISP and Scheme opposites?
10. What scoping rule is used in Scheme? In COMMON LISP? In ML? In Haskell?
11. What are three ways that ML is very different from Scheme?
12. What is type inferencing, as used in ML? (See Chapter 5.)
13. What are three features of Haskell that make it very different from Scheme?
14. What does lazy evaluation mean?

PROBLEM SET

1. Read John Backus's paper on FP (Backus, 1978) and compare the features of Scheme discussed in this chapter with the corresponding features of FP.
2. Find definitions of the Scheme functions EVAL and APPLY, and explain their actions.
3. One of the most modern and complete programming environments for Scheme is INTERLISP-ENVIRONMENT, by Teitelman and Mainsen (IEEE Computer, Vol. 14, No. 4, April 1981). Read this article carefully and compare the difficulty of writing LISP programs on your system with that of using INTERLISP (assuming that you do not normally use INTERLISP).
4. Refer to a book on LISP programming and determine what arguments support the inclusion of the PROG feature in LISP.
5. A functional language could use some data structure other than the list. For example, it could use sequences of symbols. What primitives would such a language have in place of the CAR, cdr, and CONS primitives of Scheme?
6. What does the following Scheme function do?

```scheme
(define (y s lis)
  (cond
   ((null? lis) '(1))
   ((equal? s (car lis)) lis)
   (else (y s (cdr lis)))))
)
```

7. What does the following Scheme function do?

```scheme
(define (x lis)
  (cond
   ((null? lis) 0)
   ((not (list? (car lis)))
    (cond
     ((eq? (car lis) nil) (x (cdr lis)))
     (else (+ 1 (x (cdr lis))))))
   (else (+ (x (car lis)) (x (cdr lis))))))
)
```

PROGRAMMING EXERCISES

1. Write a Scheme function that computes the volume of a sphere, given its radius.
2. Write a Scheme function that computes the real roots of a given quadratic equation. If the roots are complex, the function must display a message indicating that. This function must use an IF function. The three parameters to the function are the three coefficients of the quadratic equation.
3. Repeat Exercise 2 using a COND function, rather than an IF function.
4. Write a Scheme function that returns the number of zeros in a given simple list of numbers.
5. Write a Scheme function that deletes all top-level instances of a given atom from a given list.
6. Write a Scheme function that removes the last element from a given list.
7. Repeat Exercise 5, except that the atom can be either an atom or a list.
8. Write a Scheme function that takes two atoms and a list as parameters and replaces all occurrences of the first given atom in the list with the second given atom, no matter how deeply the first atom is nested.
9. Write a Scheme function that returns the reverse of its simple list parameter.
10. Write a Scheme predicate function that tests for the structural equality of two given lists. Two lists are structurally equal if they have the same list structure, although their atoms may be different.
11. Write a Scheme function that returns the union of two simple list parameters that represent sets.
12. Write a Scheme function with two parameters, an atom and a list, that returns the list with all occurrences, no matter how deep, of the given atom deleted. The returned list cannot contain anything in place of the deleted atoms.
13. Write a Scheme function that takes a list as a parameter and returns it with the second top-level element removed. If the given list does not have two elements, the function should return ()
14. Write a Scheme function that takes a simple list of numbers as its parameter and returns the list with the numbers in ascending order.
15. Write a Scheme function that takes a simple list of numbers as its parameter and returns the largest and smallest numbers in the list.

16. Logic Programming Languages

16.1 Introduction
16.2 A Brief Introduction to Predicate Calculus
16.3 Predicate Calculus and Proving Theorems
16.4 An Overview of Logic Programming
16.5 The Origins of Prolog
16.6 The Basic Elements of Prolog
16.7 Deficiencies of Prolog
16.8 Applications of Logic Programming