4.8 Consider the grammar:

lexp → atom | list

atom → number | identifier

list → (lexp-seq)

list-seq → list-seq lexp | lexp

(a) Remove the left recursion

Solution:

lexp -> atom | list

atom -> number | identifier

list -> (lexp-seq)

lexp-seq -> lexp lexp-seq’

lexp-seq’ -> lexp lexp-seq’ | ε

(b) Construct the First and Follow set of the nonterminals of the resulting grammar

Solution:

first (lexp)= { number ,identifier,(}

first (atom)= { number, identifier }

first (list) = { (}

first (lexp-seq)= { number ,identifier,(}

first (lexp-seq')= { number ,identifier,(, ε}

follow(lexp)= { $,),number,identifier,(}

follow (atom)= { $,), number,identifier,(}

follow (list) = {$,), number,identifier,(}

follow (lexp-seq)= {)}

follow (lexp-seq')= {)}

d. Construct the LL(1) table for the resulting grammar.
Solution:

M[N,T]	number	identifier	(	)	$
lexp	lexp->atom	lexp->atom	lexp->list		
atom	atom->number	atom->identifier			
list			list -> (lexp-seq)		
lexp-seq	lexp-seq -> lexp lexp-seq’	lexp-seq -> lexp lexp-seq’	lexp-seq -> lexp lexp-seq’		
lexp-seq’	lexp-seq’ -> lexp lexp-seq’	lexp-seq’ -> lexp lexp-seq’	lexp-seq’ -> lexp lexp-seq’	lexp-seq’ -> ε	

4.12 a. Can an LL(1) grammar be ambiguous? Why or why not?

b. Can an ambiguous be LL(1)? Why or why not?

c. Must an ambiguous be LL(1)? Why or why not?

Answer:

a. There is only one production in each entry in the parsing table, so it is not ambiguous.

b. The parsing table of an ambiguous grammar contains at least an entry with more than one production, so it cannot be LL(1).

c. No, many reasons can fail to make a grammar to be LL(1). Some unambiguous left-recursive grammar is not LL(1).