4.8

Consider the grammar

lexp→atom|list

atom→number|***identifier***

list→(lexp-seq)

lexp-seq→lexp-seq lexp|lexp

1. Remove the left recursion.

lexp→atom | list

atom→number | ***identifier***

list→(lexp-seq)

lexp-seq→lexp lexp-seq’

lexp-seq’ →lexp lexp-seq’ | ε

1. Construct First and Follow sets for the nonterminals of the resulting grammar.

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’) = {)}

1. Show that the resulting grammar is LL(1).

For every production：

First (atom) ∩ First (list) = Ø

First (number) ∩ First (identifier) = Ø

First (lexp lexp-seq’) ∩ First (ε) = Ø

For every non-terminal,

First (lexp-seq’) ∩Follow (lexp-seq’) = Ø

1. Construct the LL(1) parsing table for the resulting grammar.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |  |

1. Show the actions of the corresponding LL(1) parser, given the input string(a (b (2)) (c)).

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Parsing | Input | Action |
| 1 | $lexp | (a(b(2))(c))$ | lexp→list |
| 2 | $list | (a(b(2))(c))$ | list→(lexp-seq) |
| 3 | $)lexp-seq( | (a(b(2))(c))$ | match |
| 4 | $)lexp-seq | a(b(2))(c))$ | lexp-seq→lexp lexp-seq’ |
| 5 | $)lexp-seq’ lexp | a(b(2))(c))$ | lexp→atom |
| 6 | $)lexp-seq’ atom | a(b(2))(c))$ | atom→identifier |
| 7 | $)lexp-seq’ identifier | a(b(2))(c))$ | match |
| 8 | $)lexp-seq’ | (b(2))(c))$ | lexp-seq’ →lexp lexp-seq’ |
| 9 | $)lexp-seq’ lexp | (b(2))(c))$ | lexp→list |
| 10 | $)lexp-seq’ list | (b(2))(c))$ | list→(lexp-seq) |
| 11 | $)lexp-seq’ )lexp-seq( | (b(2))(c))$ | match |
| 12 | $)lexp-seq’ )lexp-seq | b(2))(c))$ | lexp-seq→lexp lexp-seq’ |
| 13 | $)lexp-seq’ )lexp-seq’ lexp | b(2))(c))$ | lexp→atom |
| 14 | $)lexp-seq’ )lexp-seq’ atom | b(2))(c))$ | atom→identifier |
| 15 | $)lexp-seq’ )lexp-seq’ identifier | b(2))(c))$ | match |
| 16 | $)lexp-seq’ )lexp-seq’ | (2))(c))$ | lexp-seq’ →lexp lexp-seq’ |
| 17 | $)lexp-seq’ )lexp-seq’ lexp | (2))(c))$ | lexp→list |
| 18 | $)lexp-seq’ )lexp-seq’ list | (2))(c))$ | list→(lexp-seq) |
| 19 | $)lexp-seq’)lexp-seq’ )lexp-seq( | (2))(c))$ | match |
| 20 | $)lexp-seq’)lexp-seq’ )lexp-seq | 2))(c))$ | lexp-seq→lexp lexp-seq’ |
| 21 | $)lexp-seq’)lexp-seq’)lexp-seq’lexp | 2))(c))$ | lexp→atom |
| 22 | $)lexp-seq’)lexp-seq’)lexp-seq’atom | 2))(c))$ | atom→identifier |
| 23 | $)lexp-seq’)lexp-seq’)lexp-seq’ identifier | 2))(c))$ | match |
| 24 | $)lexp-seq’)lexp-seq’)lexp-seq’ | ))(c))$ | lexp-seq’ →ε |
| 25 | $)lexp-seq’)lexp-seq’) | ))(c))$ | match |
| 26 | $)lexp-seq’)lexp-seq’ | )(c))$ | lexp-seq’ →ε |
| 27 | $)lexp-seq’) | )(c))$ | match |
| 28 | $)lexp-seq’ | (c))$ | lexp-seq’ →lexp lexp-seq’ |
| 29 | $)lexp-seq’lexp | (c))$ | lexp→list |
| 30 | $)lexp-seq’list | (c))$ | list→(lexp-seq) |
| 31 | $)lexp-seq’)lexp-seq( | (c))$ | match |
| 32 | $)lexp-seq’)lexp-seq | c))$ | lexp-seq→lexp lexp-seq’ |
| 33 | $)lexp-seq’) lexp-seq’lexp | c))$ | lexp→atom |
| 34 | $)lexp-seq’) lexp-seq’atom | c))$ | atom→identifier |
| 35 | $)lexp-seq’) lexp-seq’ identifier | c))$ | match |
| 36 | $)lexp-seq’) lexp-seq’ | ))$ | lexp-seq’ →ε |
| 37 | $)lexp-seq’) | ))$ | match |
| 38 | $)lexp-seq’ | )$ | lexp-seq’ →ε |
| 39 | $) | )$ | match |
| 40 | $ | $ | accept |

4.12 Questions:

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

LL(1)文法不会有二义性，因为它的分析表的每个入口和每个出口的产生式是唯一的。

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

二义性文法不可能是LL(1)，否则在LL(1)的分析表中会产生各种冲突。

1. Must an unambiguous grammar be LL(1)? Why or why not?

非二义的文法不一定是LL(1)，因为二义文法是非LL(1)文法的一个因素，但不是唯一的因素，如一些带有左递归的文法也不是 LL(1)文法，因为左递归文法会让它的递归下降语法分析器进入一个无限循环。