Compilation Principle Homework 3

Jinyan XU, 3160101126, Information Security

4.8 Consider the grammar:

*lexp* → *atom* | *list*  **①**

*atom* →**number**|**identifier ②**

*list* →( *lexp-seq* )  **③**

*lexp-seq* → *lexp-seq lexp* | *lexp* **④**

(a) Remove the left recursion

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

(c) Show the grammar is LL(1).

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

(e) Show the actions of the corresponding parser, given the following input string: **(a (b (2)) (c))**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Answer：**   1. Only **④** is left recursion, so:   *lexp* → *atom* | *list* **①**  *atom* → **number** | **identifier** **②**  *list* → ( *lexp-seq* ) **③**  *lexp-seq* → *lexp lexp-seq’* **④**  *lexp-seq’* → *lexp lexp-seq’* | ε **⑤**   1. 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’) = { ) }  c）This grammar is LL(1), because:  For the productions have same left side, **① ② ⑤**, First(atom) First(list) = First(number) First(identifier) = First(lexp lexp-seq’) First(ε) = Φ  For the First Set who has ε, First(lexp-seq’)  Follow(lexp-seq’) = Φ  Besides, we can also find that in parsing table **↓**, each unit only has one production.  d)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 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 |  |   e)   |  |  |  | | --- | --- | --- | | Parsing Stack | Input String | Action | | $ lexp | (a(b(2))(c))$ | lexp → list | | $ list | (a(b(2))(c))$ | list → ( lexp-seq ) | | $) lexp-seq ( | (a(b(2))(c))$ | match | | $) lexp-seq | a(b(2))(c))$ | lexp-seq → lexp lexp-seq’ | | $) lexp-sep' lexp | a(b(2))(c))$ | lexp → atom | | $) lexp-sep' atom | a(b(2))(c))$ | atom → identifier | | $) lexp-sep' identifier | a(b(2))(c))$ | match | | $) lexp-sep' | (b(2))(c))$ | lexp-seq’ → lexp lexp-seq’ | | $) lexp-seq' lexp | (b(2))(c))$ | lexp → list | | $) lexp-seq' list | (b(2))(c))$ | list → ( lexp-seq ) | | $) lexp-sep' ) lexp-seq ( | (b(2))(c))$ | match | | $) lexp-sep' ) lexp-seq | b(2))(c))$ | lexp-seq → lexp lexp-seq’ | | $) lexp-sep' ) lexp-seq' lexp | b(2))(c))$ | lexp → atom | | $) lexp-sep' ) lexp-seq' atom | b(2))(c))$ | atom → identifier | | $) lexp-sep' ) lexp-seq' identifier | b(2))(c))$ | match | | $) lexp-sep' ) lexp-seq' | (2))(c))$ | lexp-seq’ → lexp lexp-seq’ | | $) lexp-sep' ) lexp-seq' lexp | (2))(c))$ | lexp → list | | $) lexp-sep' ) lexp-seq' list | (2))(c))$ | list → ( lexp-seq ) | | $) lexp-sep' ) lexp-seq' ) lexp-seq ( | (2))(c))$ | match | | $) lexp-sep' ) lexp-seq' ) lexp-seq | 2))(c))$ | lexp-seq → lexp lexp-seq’ | | $) lexp-sep' ) lexp-seq' ) lexp-seq' lexp | 2))(c))$ | lexp → atom | | $) lexp-sep' ) lexp-seq' ) lexp-seq' atom | 2))(c))$ | atom → number | | $) lexp-sep' ) lexp-seq' ) lexp-seq' number | 2))(c))$ | match | | $) lexp-sep' ) lexp-seq' ) lexp-seq' | ))(c))$ | lexp-seq’ → ε | | $) lexp-sep' ) lexp-seq' ) | ))(c))$ | match | | $) lexp-sep' ) lexp-seq' | )(c))$ | lexp-seq’ → ε | | $) lexp-sep' ) | )(c))$ | match | | $) lexp-sep' | (c))$ | lexp-seq’ → lexp lexp-seq’ | | $) lexp-sep' lexp | (c))$ | lexp → list | | $) lexp-sep' list | (c))$ | list → ( lexp-seq ) | | $) lexp-sep' ) lexp-seq ( | (c))$ | match | | $) lexp-sep' ) lexp-seq | c))$ | lexp-seq → lexp lexp-seq’ | | $) lexp-sep' ) lexp-seq' lexp | c))$ | lexp → atom | | $) lexp-sep' ) lexp-seq' atom | c))$ | atom → identifier | | $) lexp-sep' ) lexp-seq' identifier | c))$ | match | | $) lexp-sep' ) lexp-seq' | ))$ | lexp-seq’ → ε | | $) lexp-sep' ) | ))$ | match | | $) lexp-sep' | )$ | lexp-seq’ → ε | | $) | )$ | match | | $ | $ | accept | |

4.12 Answer the question:

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

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

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

|  |
| --- |
| **Answer：**   1. No, in LL(1) parsing table, each unit only has one production, it can’t be ambiguous. 2. No, an ambiguous grammar means unit will have more than one productions, which is not LL(1). 3. No, if the unambiguous grammar has left recursion productions, it can’t be LL(1) either. |