Compilation Principle Homework 2

Jinyan XU, 3160101126, Information Security

3.2 Given the grammar A → AA | (A) | ε,

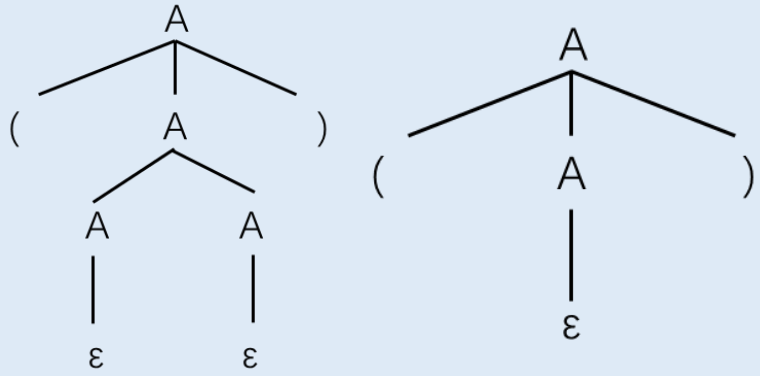
a. Describe the language it generates.

b. Show it is ambiguous.

Answer:

a. It generates a sequence of symmetric parentheses, you can find a pair of parentheses for each one, and this language contains the empty string.

b. Ambiguous can be like this:



The string "()" can generate two different parse tree, so this language is ambiguous.

3.3 Given the grammar:

exp → exp addop term | term
addop → + | -
term → term mulop factor | factor
mulop → *
factor → (exp) | factor

Write down leftmost derivation, parse trees, and absctract syntax trees for the following expressions:

(a) 3+4*5-6

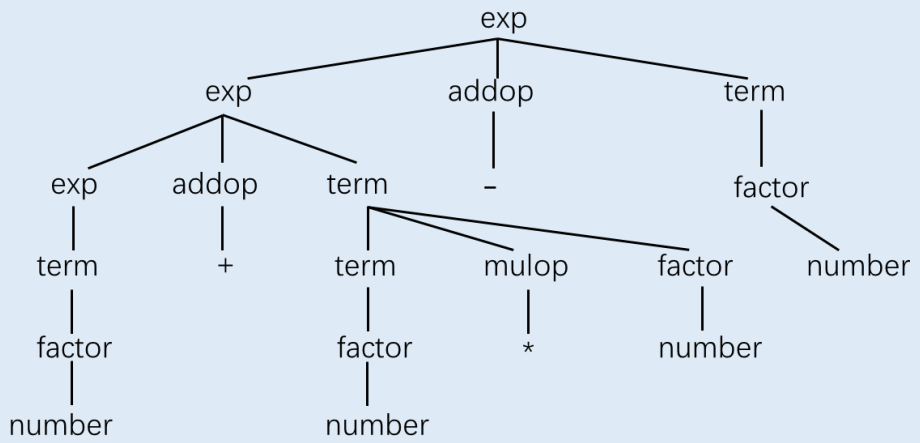
Answer:

Leftmost derivation:

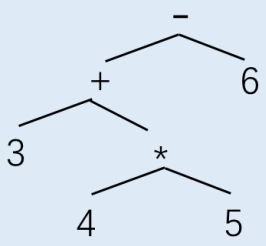
```
exp => exp addop term
    => exp addop term addop term
    => term addop term addop term
    => factor addop term addop term
    => number addop term addop term
    => number + term addop term
    => number + term mulop factor addop term
    => number + factor mulop factor addop term
    => number + number mulop factor addop term
    => number + number * factor addop term
    => number + number * number addop term
```

```
    => number + number * number - term
    => number + number * number - factor
    => number + number * number - number
```

Parse Tree:



Absctract Syntax Tree:



3.4 The following grammar generates all regular expressions over the alphabet of letters (we have to use quotes to surround operators, since the vertical bar is an operator as well as a metasymbol):

rexp → rexp "|" rexp
| rexp rexp
| rexp "*"
| "(" rexp ")"
| letter

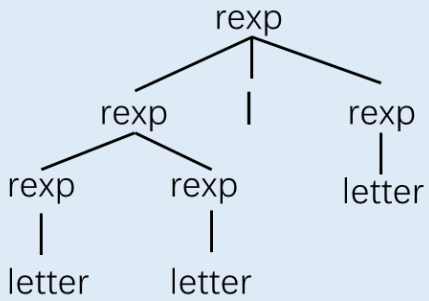
a. Give a derivation for the regular expression (ab | a)* using this grammar.
b. Show that this grammar is ambiguous.
c. Rewrite this grammar to establish the correct precedences for the operations (see chapter 2).
d. What associativity does your answer in part (c) give to the binary operations? Why?

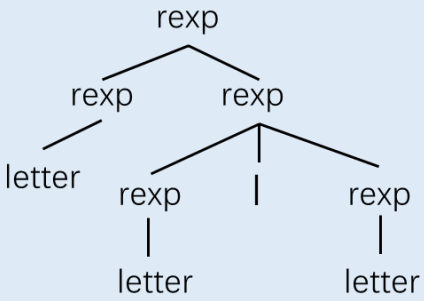
Answer:

a.

```
rexp   => rexp "*"
       => "(" rexp ")" "*"
       => "(" rexp "|" rexp ")" "*"
       => "(" rexp rexp "|" rexp ")" "*"
       =>* "(" letter letter "|" letter ")" "*"
```

b. Ambiguous can be like this:





The string “ab | c” can generate two different parse tree, so this language is ambiguous.

c. rexp → rexp “|” rexp_conc | rexp_conc
   rexp_conc → rexp_conc rexp_clos | rexp_clos
   rexp_ clos → rexp_clos “*” | rexp_pare
   rexp_ pare → “(” rexp “)” | letter

d. Because I use a left recursive in the production, so its operators associate on the left. If not, string like “abc” can have 2 kinds parse trees. But in fact, it’s a inessential ambiguity, I mean that (ab)c = a(bc), or (a|b)|c = a|(b|c), their syntax trees are still distinct, but semantic value are the same.