1. Consider the context-free grammar:

S -> S S + | S S * | a

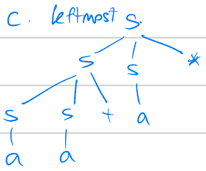
and the string aa + a*.

   a. Give a leftmost derivation for the string.
   b. Give a rightmost derivation for the string.
   c. Give a parse tree for the string.
   d. Is the grammar ambiguous or unambiguous? Justify your answer.

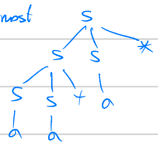
a. S → SS*<br>
→ SS+S*<br>
→ aS+S*<br>
→ aa + S*<br>
→ aa + a*

b. S → SS*<br>
→ Sa*<br>
→ SS+a*<br>
→ Sa +a*<br>
→ aa + a*

c. leftmost



rightmost



d. unambiguous, Leftmost and rightmost derivatives give the same parse tree.

2. Consider the context-free grammar:

S -> S + S | S S | (S) | S * | a

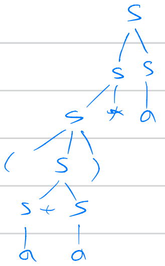
and the string (a+a)*a.

   a. Give a leftmost derivation for the string.
   b. Give a rightmost derivation for the string.
   c. Give a parse tree for the string.
   d. Is the grammar ambiguous or unambiguous? Justify your answer.

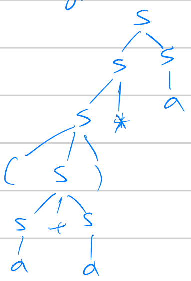
a. S → SS<br>
→ S*S<br>
→ (S)*S<br>
→ (S+S)*S<br>
→ (a+S)*S<br>
→ (a+a)*S<br>
→ (a+a)*a

b. S → SS<br>
→ Sa<br>
→ S*a<br>
→ (S)*a<br>
→ (S+S)*a<br>
→ (S+a )*a<br>
→ (a+a)*a

c. Left



right



d. unambiguous, Leftmost and rightmost derivatives give the same parse tree.

3. Design grammars for the following languages:

   a. The set of all strings of 0s and 1s such that every 0 is immediately followed by at least one 1.
   b. The set of all strings of 0s and 1s that are palindromes; that is, the string reads the same backward as forward.
   c. The set of all strings of 0s and 1s with an equal number of 0s and 1s.
   d. The set of all strings of 0s and as in which 011 does not appear as a substring.

a. S → (0?1)* or S → AB<br>
A → 1A | ε 1*<br>
B → CB | ε (011*)*<br>
C → 01A 011*

b. S → 0S0 | 1S1 | 0 | 1 | ε

c. S → 0S1S | 1S0S | ε

d. S → 1*(0+1?)*

d. The set of all strings of 0s and as in which 011 does not appear as a substring.

4. The following is a grammar for regular expressions over symbols a and b only, using + in place of | for union, to avoid conflict with the use of vertical bar as a metasymbol in grammars:

```
rexpr -> rexpr + rterm | rterm
rterm -> rterm rfactor | rfactor
rfactor -> rfactor * | rprimary
rprimary -> a | b
```

   a. Left factor this grammar.
   b. Does left factoring make the grammar suitable for top-down parsing?
   c. In addition to left factoring, eliminate left recursion from the original grammar.
   d. Is the resulting grammar suitable for top-down parsing? Justify your answer.

a. Cannot do left factor

b. not suitable

c. rexpr → rterm A<br>
A → +rterm A | ε<br>
rterm → rfactor B | ε<br>
B → rfactor B | ε<br>
rfactor → rprimary C<br>
C → *C | ε<br>
rprimary → a | b

d. suitable

## Exercise 2

1. Consider the context-free grammar:

   S -> S S + | S S * | a

   and the string aa + a*.

   a. Give a leftmost derivation for the string.

   b. Give a rightmost derivation for the string.

   c. Give a parse tree for the string.

   d. Is the grammar ambiguous or unambiguous? Justify your answer.

2. Consider the context-free grammar:

   S -> S + S | S S | (S) | S * | a

   and the string (a+a)*a.

   a. Give a leftmost derivation for the string.

   b. Give a rightmost derivation for the string.

   c. Give a parse tree for the string.

   d. Is the grammar ambiguous or unambiguous? Justify your answer.

3. Design grammars for the following languages:

   a. The set of all strings of 0s and 1s such that every 0 is immediately followed by at least one 1.

   b. The set of all strings of 0s and 1s that are palindromes; that is, the string reads the same backward as forward.

   c. The set of all strings of 0s and 1s with an equal number of 0s and 1s.

   d. The set of all strings of 0s and as in which 011 does not appear as a substring.

4. The following is a grammar for regular expressions over symbols a and b only, using + in place of | for union, to avoid conflict with the use of vertical bar as a metasymbol in grammars:

   ```
   rexpr -> rexpr + rterm | rterm
   rterm -> rterm rfactor | rfactor
   rfactor -> rfactor * | rprimary
   rprimary -> a | b
   ```

   a. Left factor this grammar.

   b. Does left factoring make the grammar suitable for top-down parsing?

   c. In addition to left factoring, eliminate left recursion from the original grammar.

   d. Is the resulting grammar suitable for top-down parsing? Justify your answer.