Nawal Ahmed
Concepts Programming Languages
Assignment #2

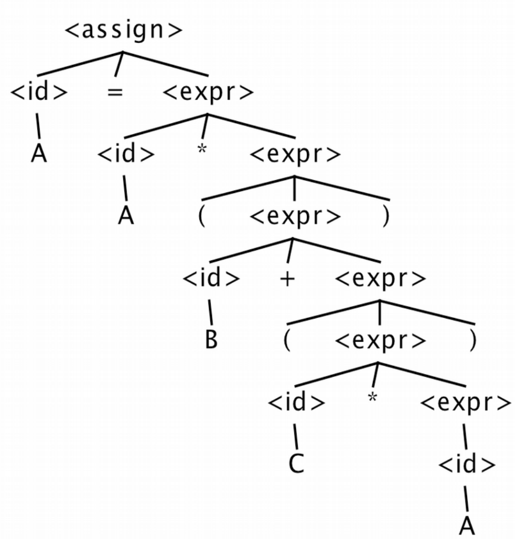
1. Write a grammar for the language consisting of strings that have *n* copies of the letter a followed by the same number of copies of the letter b, where *n* > 0. For example, the strings ab, aaabbb, and aaaaabbbbb are in the language, but a, abb, ba, and aaabb are not in the language.

   Solution: <S> → ab | a <S>b

2. Using the grammar below, show a parse tree and a leftmost derivation for each of the following statements:

   <assign> -> <id> = <expr>
   <id> -> A | B | C
   <expr> -> <id> + <expr>
   | <id> * <expr>
   | (<expr>)
   | <id>

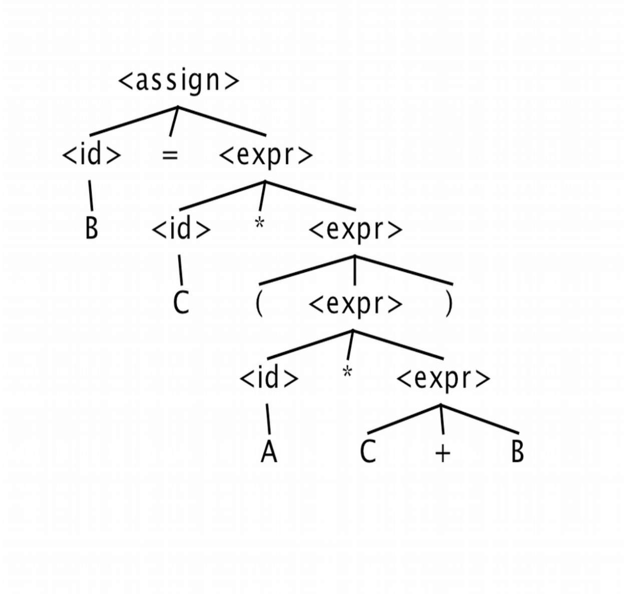
   a. A = A * (B + (C * A))



Solution

<assign> → <id> = <expr>
→ A = <expr>
→ A = <id> * <expr>
→ A = A * <expr>
→ A = A * (<expr>)
→ A = A * (<id> + <expr>)
→ A = A * (B + <expr>)
→ A = A * (B + (<expr>))
→ A = A * (B + (<id> * <expr>))
→ A = A * (B + (C * <expr>))
→ A = A * (B + (C * <id>))
→ A = A * (B + (C * A))

b. B = C * (A * C + B)



Solution

<assign> → id = <expr>
→ B = <expr>
→ B = id * <expr>
→ B = C * <expr>
→ B = C * (<expr>)
→ B = C * (id * <expr>)
→ B = C * (A * <expr>)
→ B = C * (A * id + <expr>)
→ B = C * (A * C + <expr>)
→ B = C * (A * C * id)
→ B = C * (A * C + B)

3. Prove the following grammar is ambiguous:

<assign> -> <id> = <expr>
<id> -> A | B | C
<expr> -> <expr> + <expr>
| <expr> * <expr>
| (<expr>)
| <id>

I can prove a grammar is ambiguous by creating two leftmost derivations for the same sentence. Consider the sentence, A = B + C * A. Below are two different leftmost derivations for the same sentence.

4. Consider the following grammar:

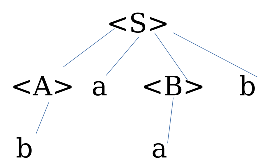
<S> -> <A> a <B> b
<A> -> <A> b | b
<B> -> a <B> | a

Which of the following sentences are in the language generated by this grammar?

a. baab
b. bbbab
c. bbaaaaaS
d. bbaab

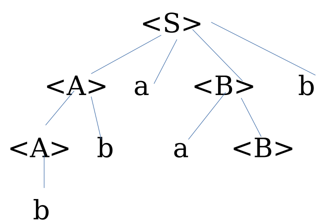
Solution:

a.



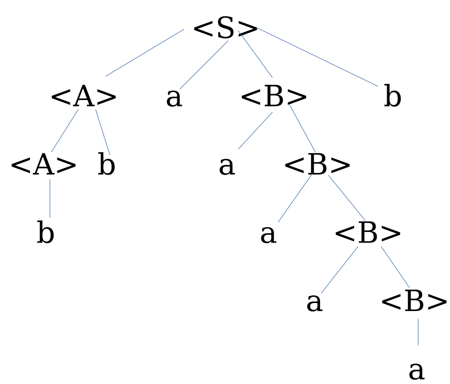
The sentence is generated.

b.



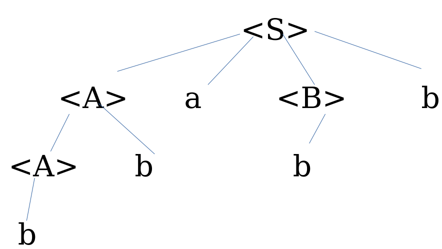
The sentence is not generated.

c.



The sentence is not generated.

d.



The sentence is generated.

5. Convert the following EBNF to BNF:

<S> -> <A> { b<A> }
<A> -> a[b]<A>

Solution:

S → A | A B
B → b A | b A B
A → a A | a b A