Aleksandr Itskovich

Sebesta Chapter 3

*3. Rewrite the BNF example 3.4 to give + precedence over * and force + to be right associative.*

<assign> -> <id> = <expr>

<id> -> A | B | C

<expr> -> <expr> * <term> | <term>

<term> -> <factor > + <term> | <factor>

<factor> -> (<expr>) | <id>

4.Rewrite the BNF of example 3.4 to add the ++ and -- unary operators of Java.

<assign> -> <id> = <expr>

<id> = A | B | C.

<expr> -> <expr> + <term> | <term>

<term> -> <term> * <factor> | <factor>

<factor> -> (<expr>) | <term> ++ | <term> -- | ++<term> | -- <term> | <id>

6. Using the grammar in Example 3.2, show a parse tree and a leftmost derivation for each of the following statements:

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

<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)*

<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)

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

<assign> => <id> = <expr>

=> A = <expr>

=> A = <id> * <expr>

=> A = A * <expr>

=> A = A * (<expr>)

=> A = A * (<id> + <expr>)

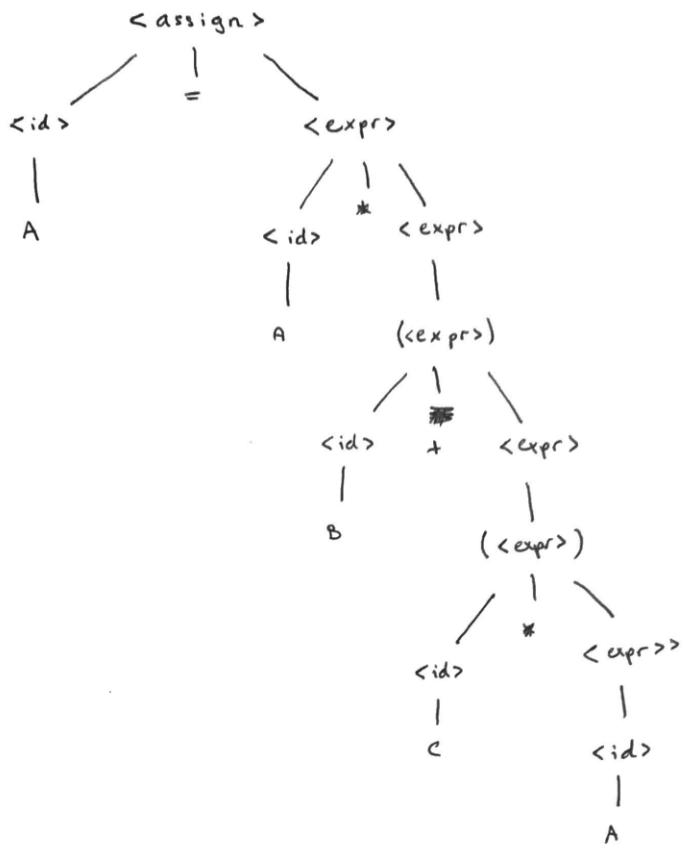
=> A = A * (B + <expr>)

=> A = A * (B + (<expr>))

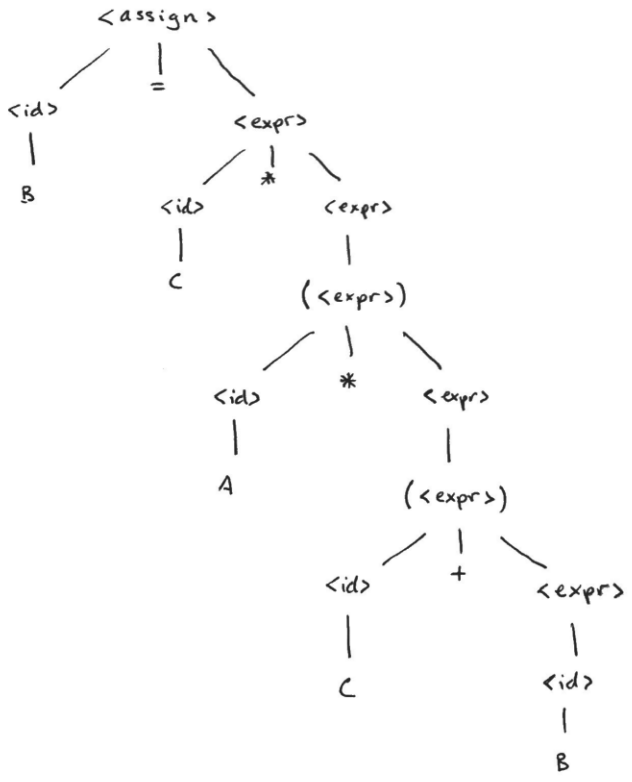
=> A = A * (B + (<id>))

=> A = A * (B + (C))

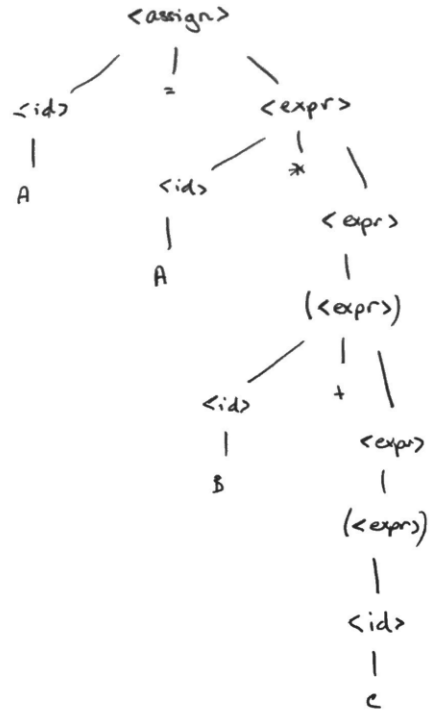
a.


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

b.


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

c.


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

7.

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

<assign> => <id> = <expr>

=> A = <expr>

=> A = <term>

=> A = <term> * <factor>

=> A = <factor> * <factor>

=> A = (<expr>) * <factor>

=> A = (<expr> + <term>) * <factor>

=> A = (<term> + <term>) * <factor>

=> A = (<factor> + <term>) * <factor>

=> A = (<id> + <term>) * <factor>

=> A = (A + <term>) * <factor>

=> A = (A + <factor>) * <factor>

=> A = (A + <id>) * <factor>

=> A = (A + B) * <factor>

=> A = (A + B) * <id>

=> A = (A + B) * C

b. A = B + C + A

<assign> => <id> = <expr>

=> A = <expr>

=> A = <expr> + <term>

=> A = <term> + <term>

=> A = <factor> + <term>

=> A = <id> + <term>

=> A = B + <term>

=> A = B + <factor>

=> A = B + <expr>

=> A = B + <expr> + <term>

=> A = B + <term> + <term>

=> A = B + <factor> + <term>

=> A = B + <id> + <term>

=> A = B + C + <term>

=> A = B + C + <factor>

=> A = B + C + <id>

=> A = B + C + A

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

<assign> => <id> = <expr>

=> A = <expr>

=> A = <term>

=> A = <term> * <factor>

=> A = <factor> * <factor>

=> A = <id> * <factor>

=> A = A * <factor>

=> A = A * (<expr>)

=> A = A * (<expr> + <term>)

=> A = A * (<term> + <term>)

=> A = A * (<factor> + <term>)

=> A = A * (<id> + <term>)

=> A = A * (B + <term>)

=> A = A * (B + <factor>)

=> A = A * (B + <id>)

=> A = A * (B + C)

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

<assign> => <id> = <expr>

=> A = <expr>

=> A = <term>

=> A = <term> * <factor>

=> A = <factor> * <factor>

=> A = <id> * <factor>

=> A = B * <factor>

=> A = B * (<expr>)

=> A = B * (<term>)

=> A = B * (<term> * <factor>)

=> A = B * (<factor> * <factor>)

=> A = B * (<id> * <factor>)

=> A = B * (C * <factor>)

=> A = B * (C * (<expr>))

=> A = B * (C * (<expr> + <term>))

=> A = B * (C * (<term> + <term>))

=> A = B * (C * (<factor> + <term>))

=> A = B * (C * (<id> + <term>))

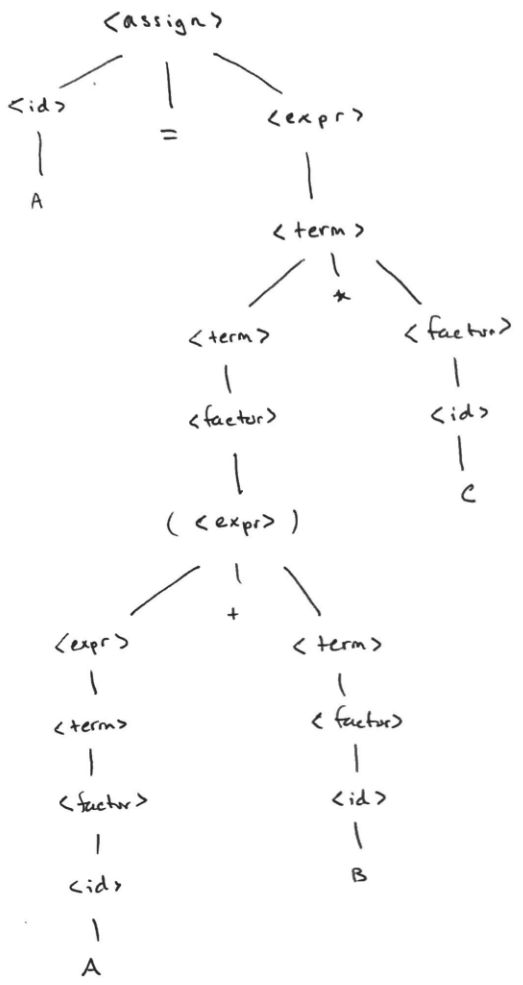
=> A = B * (C * (A + <term>))

=> A = B * (C * (A + <factor>))

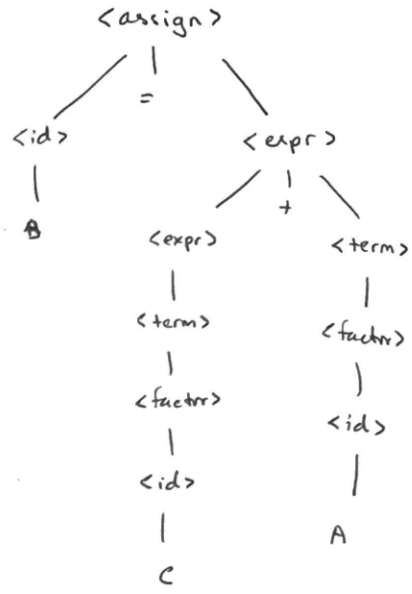
=> A = B * (C * (A + <id>))

=> A = B * (C * (A + B))

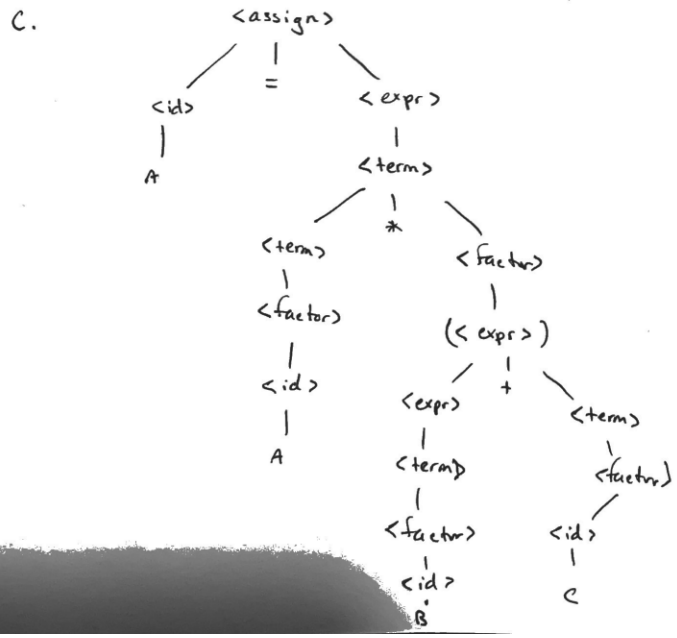
a.


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

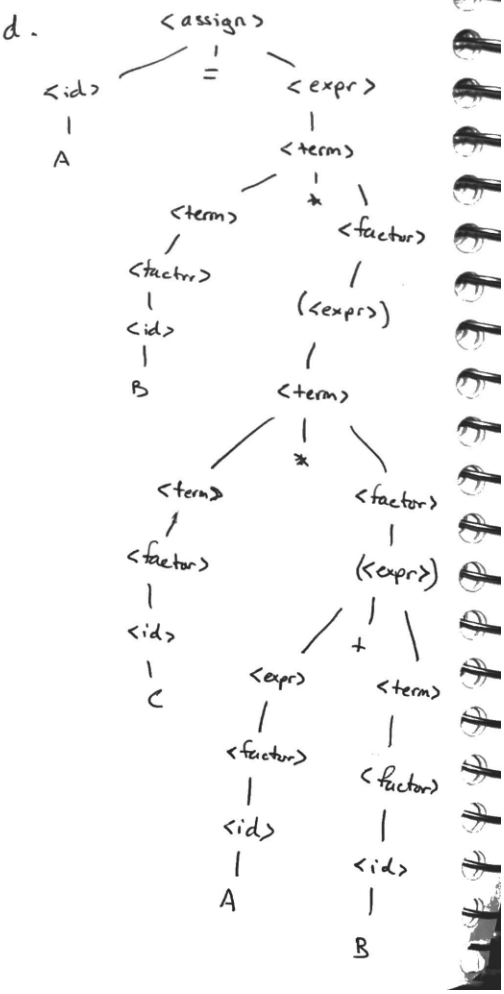
b.


<assign>
<id>
=
<expr>
B
<expr>
+
<term>
<term>
<factor>
<factor>
<id>
<id>
C
A

c.


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

d.


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

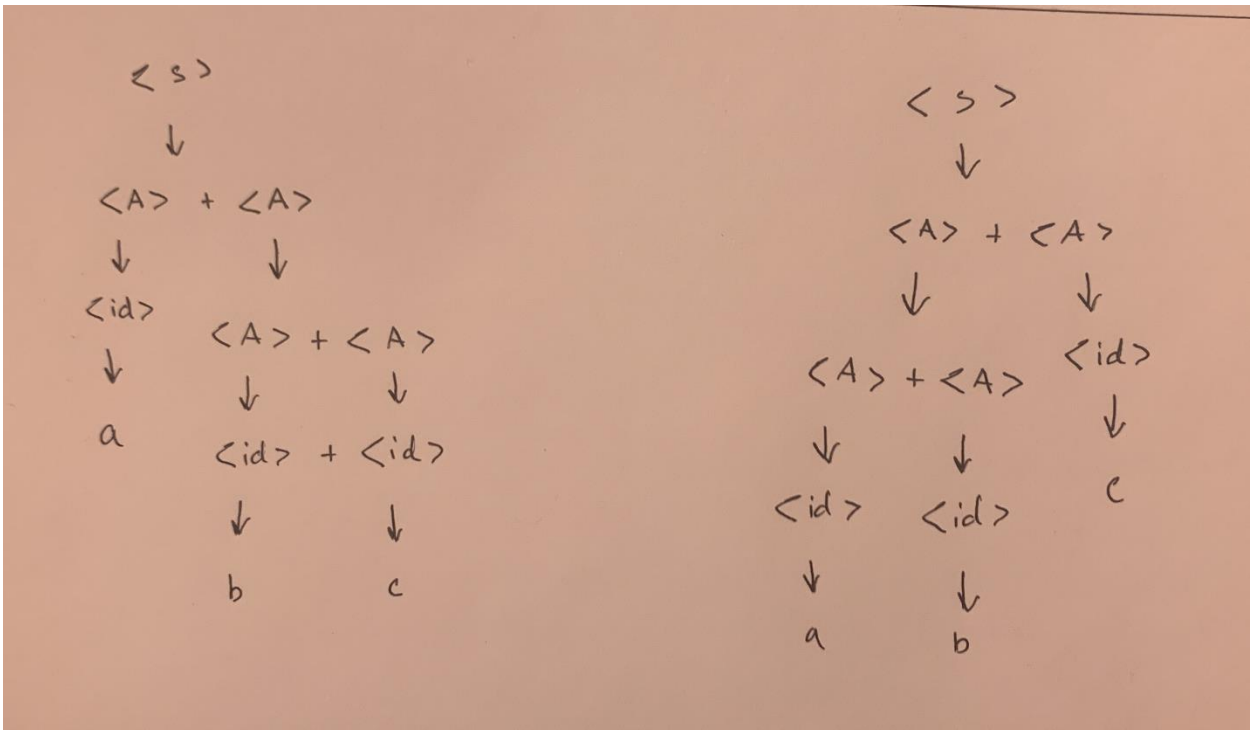
8. Prove that the following grammar is ambiguous:

<S> --> <A>

<A> --> <A> + <A> | <id>

<id> --> a | b | c

This grammar is ambiguous because the same sentence can be produced by two or more different parse trees. For example, the sentence “a + b + c” can be produced by:



11. 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 (Yes)

b. bbbab (No)

c. bbaaaaa (No)

d. bbaab (Yes)

12. Consider the following grammar:

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

<A> --> c <A> | c

<B> --> d | <A>

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

a. abcd (Yes)

b. acccbd (No)

c. acccbcc (No)

d. acd (No)

e. accc (Yes)

13. 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, aaaabbbb, and aaaaaaaabbbbbbbb are in the language, but a, abb, ba and aaabb are not.

<S> --> a <S> b | ab

14. Draw parse trees for the sentences aabb and aaaabbbb, as derived from the grammar of problem 13.