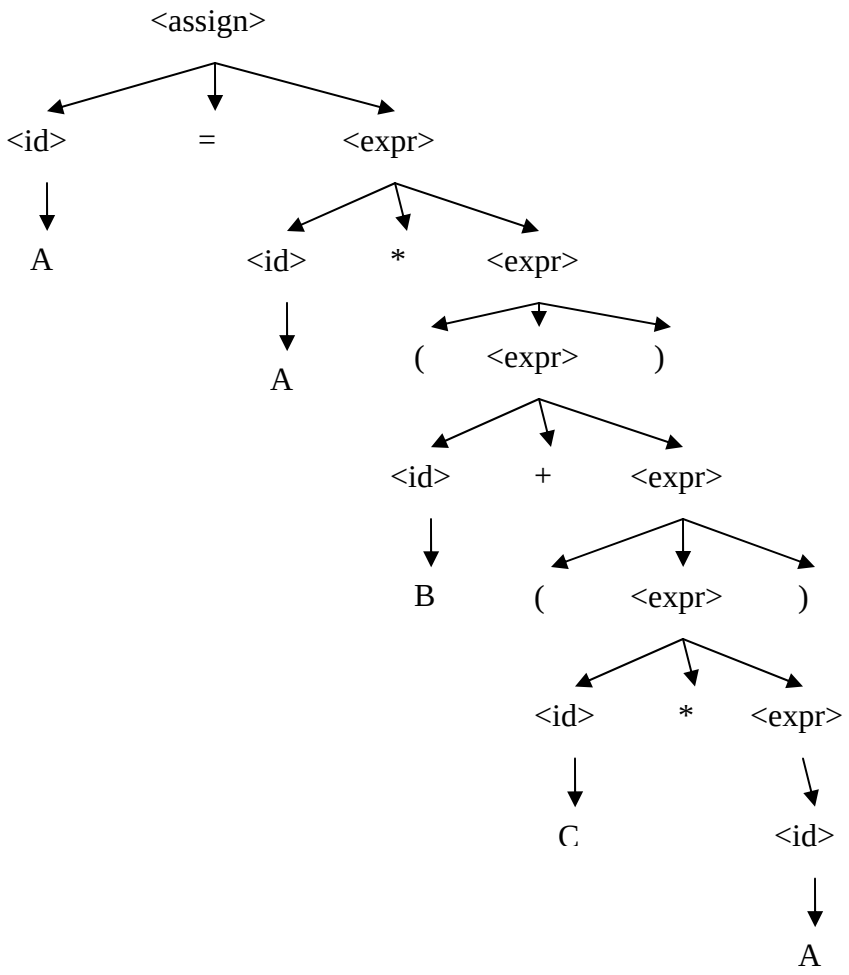
1.

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

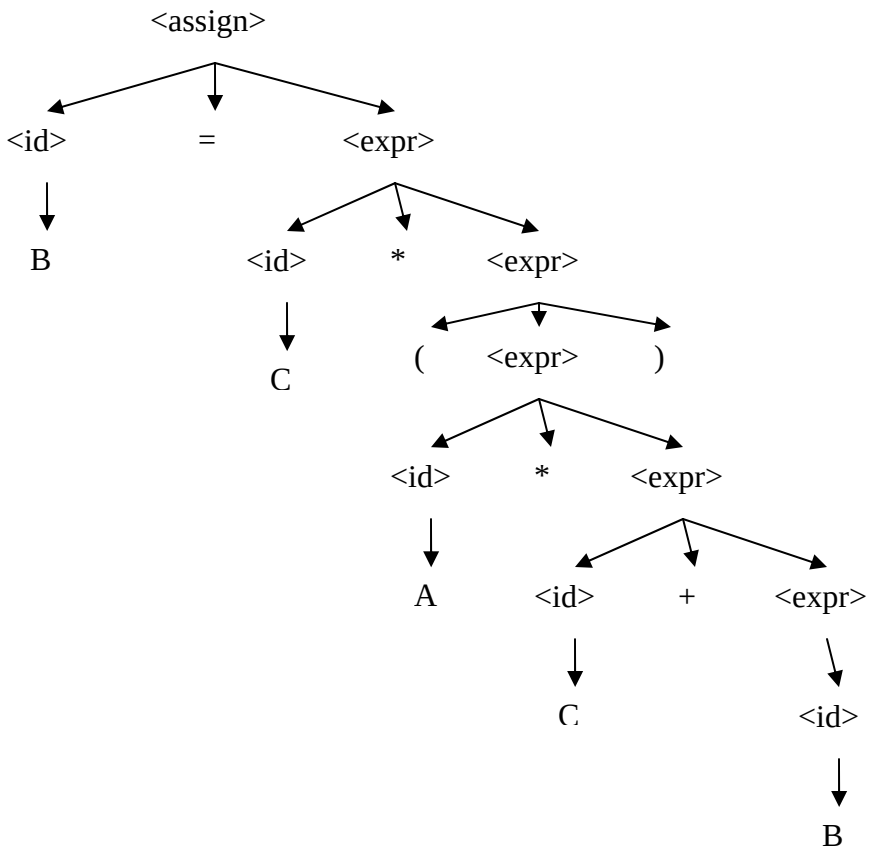
2.

(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))

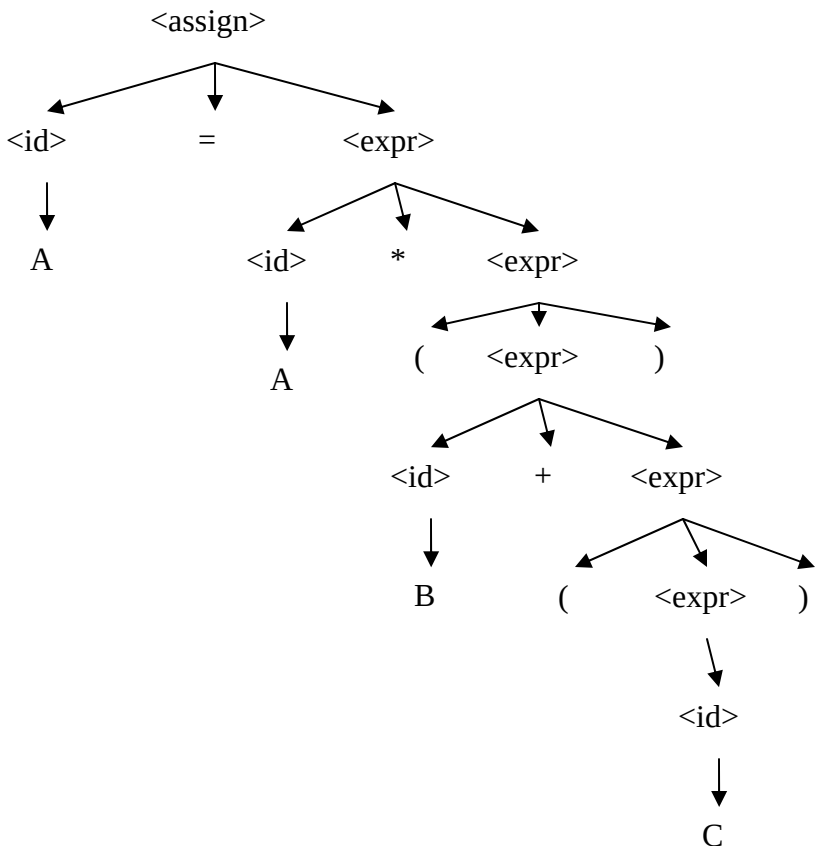


(b)

<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)

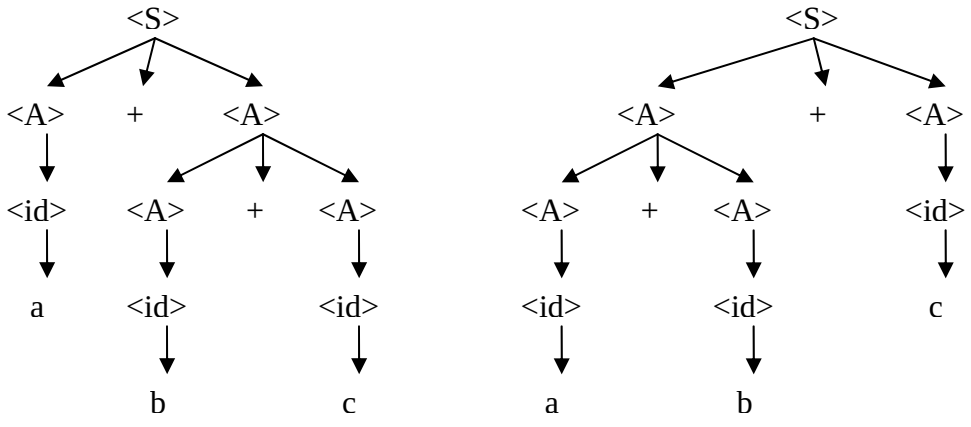


(c)
<assign> => <id> = <expr>
=> A = <id> * <expr>
=> A = A * <expr>
=> A = A * (<expr>)
=> A = A * (<id> + <expr>)
=> A = A * (B + <expr>)
=> A = A * (B + (<expr>))
=> A = A * (B + (<id>))
=> A = A * (B + (C))



3.

Given “abc”, two parsing trees are legal



or

So, the grammar is ambiguous.

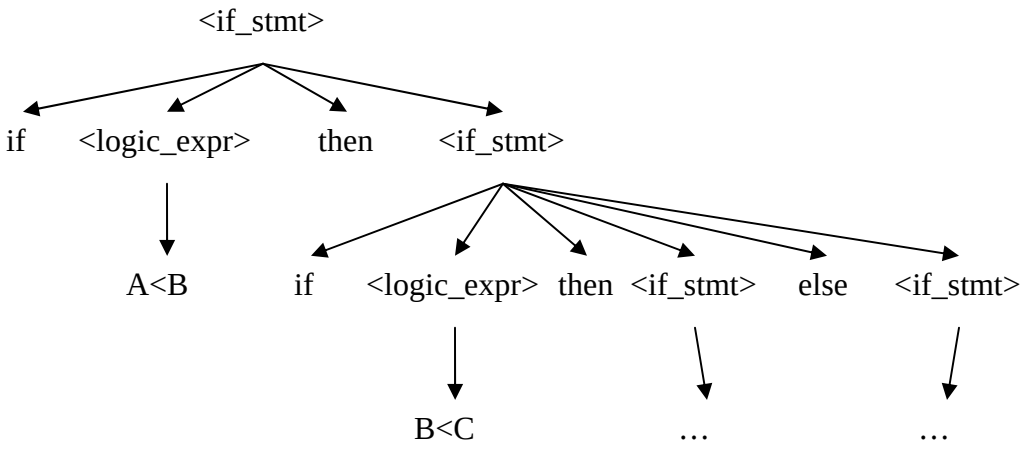
4.

<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <term> | <term>
<term> → <term> * <factor> | <factor>
<factor> → (<expr>) | <minus>
<minus> → - <id> | <id>

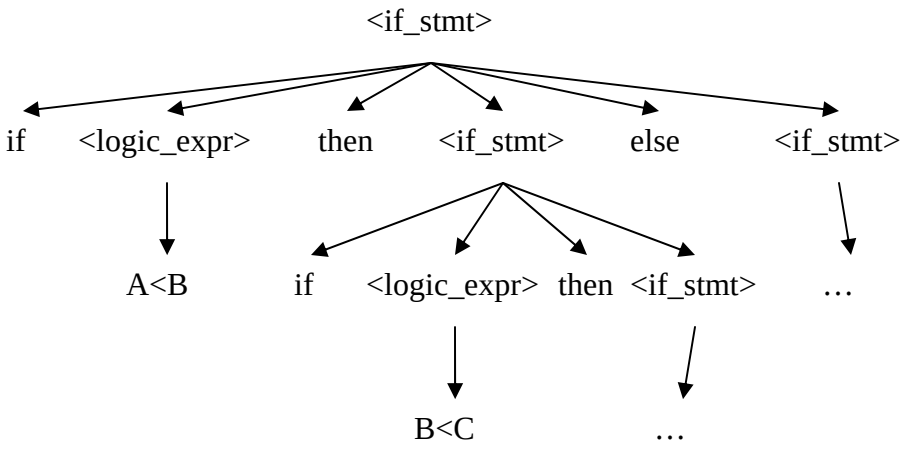
5.

Consider following snippet
If A<B then *dosth1*
If B<C then *dosth2*
Else

Two parsing trees are legal



Or



So, the grammar is ambiguous.

Java solves the problem by separating if-then clause into two nonterminals, “if-then” and “if-then-else” (named with a postfix “short if” in Java BNF, http://www.daimi.au.dk/dRegAut/JavaBNF.html) while strictly applying the leftmost derivation. That ensures that the “else” terminal always matches the nearest “if-then” nonterminal. Or, in other word, arbitrarily decree that an else clause belongs to the innermost if to which it might possibly belong.

The corresponding modification is

<stmt> → <if_stmt> | <if_else_stmt> | <other_stmt>
<stmt_no_short_if> → <if_else_stmt_no_short_if> | <other_stmt>
<if_stmt> → if <logic_expr> then <stmt>
<if_else_stmt> → if <logic_expr> then <stmt_no_short_if> else <stmt>
<if_else_stmt_no_short_if> → if <logic_expr> then <stmt_no_short_if> else <stmt_no_short_if>

Corresponding leftmost derivation of previous snippet

snippet => <stmt>
=> <if_stmt>
=> if <logic_expr> then <stmt>
=> if A<B then <stmt>
=> if A<B then if <logic_expr> then <stmt_no_short_if> else <stmt>
=> if A<B then if B<C then <stmt_no_short_if> else <stmt>
=> if A<B then if B<C then <other_stmt> else <other_stmt>
=> …