1. parse tree:

```
  bexpr  
   /    
  or    bterm 
   |      /  
  bexpr  bterm 
   |      |   
  bterm  bfactor 
   |       |   
  and    true  not  bfactor
   |       |   |      |        |       |       
  false  bexpr  bterm  bfactor  true
```

2. EBNF grammar:

```
bexpr  ->  bterm ( or bterm )*  
bterm  ->  bfactor ( and bfactor )*  
bfactor ->  not bfactor  
           |        "(" bexpr ")" 
           |        true  
           |        false
```

3.

(a) AST:

```
BinaryExpr  
  /    
BinaryExpr  or  UnaryExpr  
  |      |      
FalseLiteral and  TrueLiteral  not  TrueLiteral
```

(b)  

```
b1 = new FalseLiteral();
b2 = new TrueLiteral();
b3 = new BinaryExpr(b1, "and", b2);
b4 = new TrueLiteral();
b5 = new UnaryExpr("not", b4);
b6 = new BinaryExpr(b3, "or", b5);
```
4. Attribute grammar for constructing ASTs:

\[
\begin{align*}
\text{bexpr} & \rightarrow \text{bterm \ [ bexpr.node := bterm.node ]} \\
& \quad ( \text{or } \text{bterm \ [ bexpr.node := BinaryExpr(bexpr.node, "or", bterm.node) ] } )^* \\
\text{bterm} & \rightarrow \text{bfactor \ [ bterm.node := bfactor.node ]} \\
& \quad ( \text{and } \text{bfactor \ [ bterm.node := BinaryExpr(bterm.node, "and", bfactor.node) ] } )^* \\
\text{bfactor} & \rightarrow \text{not bfactor \ [ bfactor.node := UnaryExpr("not", bfactor.node) ]} \\
& \quad \text{"( } \text{bexpr } \text{")" } [ \text{factor.node := bexpr.node} ] \\
& \quad \text{true } [ \text{factor.node := TrueLiteral} ] \\
& \quad \text{false } [ \text{factor.node := FalseLiteral} ]
\end{align*}
\]

5.

By examining the parse and syntax trees for the example boolean expression, we note that separators (or punctuation tokens), such as ( and ), useful for parsing the program are redundant and convey no useful information. Therefore, separators do not need to be present in the syntax tree.

In addition, the structure of the parse tree depends on too much on the grammar. Many extra nonterminals are used for the purposes of enforcing the correct precedence and associativity of operators. These details should be confined to the parsing process.