CS558 Programming Languages – Fall 2020 – Suggested Study Question Solutions for Lecture 1b

1. This grammar is inspired by the well-known “dangling else” ambiguity problem, which arises in languages where the `else` clause of an `if-then-else` statement is optional and the `then` and `else` bodies can be arbitrary statements (including `if-then-else` statements!) For example, in the sentence of part (c), the ambiguity of the grammar means that we cannot tell which `if` the `else` should be paired up with.

   (a) ![Diagram](https://example.com/diagram.png)

(b) Here are two possible derivations; the first is the left-most derivation for the tree in (a); the second is the right-most derivation.

   - \( S \rightarrow i B t S \)
   - \( S \rightarrow i n t S \)
   - \( S \rightarrow i n t i B t S \)
   - \( S \rightarrow i n t i y t S \)
   - \( S \rightarrow i n t i y t k \)
   - \( S \rightarrow i B t S \)
   - \( S \rightarrow i B t i B t S \)
   - \( S \rightarrow i B t i B t k \)
   - \( S \rightarrow i B t i y t k \)
   - \( S \rightarrow i n t i y t k \)

(c) Here are two trees:

   ![Tree1](https://example.com/tree1.png) ![Tree2](https://example.com/tree2.png)

(d) The grammar is ambiguous because there is at least one sentence—in particular, the one in part (c)—that has two different parse trees.

Note that it is not correct to claim that the grammar is ambiguous because the same sentence has more than one linear derivation sequence. Even when a sentence has a unique parse tree, as in part (a), it may have multiple corresponding linear derivation sequences, as in part (b). Such a sentence is not enough to demonstrate ambiguity: we must find a sentence that has two different trees.
2. (a) Here’s the parse tree:

(b) We simply switch the order of the nonterminals in the right-hand side of the production for $E$:

$$E ::= T + E | T - E | T$$

The rest of the grammar remains the same. Try it on an example like $a + b + c$.

3. Here is a possible AST tree grammar:

```
Add : Exp -> Exp Exp
Sub : Exp -> Exp Exp
Mul : Exp -> Exp Exp
Div : Exp -> Exp Exp
Id : Exp -> (string)
```

(b)

```
Sub
  Mul  Id c
    Id a  Add
       Id b  Id c
```

The most important differences from a parse tree are that: (a) intermediate levels of nonterminals (such as $T$ and $F$), have disappeared, because they were only in the concrete grammar to enforce the intended precedence and associativity, which are now directly reflected in the shape of the tree; and (b) there are no explicit parentheses, again because their grouping effect is now directly reflected in the shape of the tree.