- Disambiguate the following grammar:

$$E \rightarrow E + E \mid E - E \mid E \times E \mid (E) \mid V$$
$$V \rightarrow x \mid y \mid z$$

Ans: To disambiguate the given grammar, we need to address the ambiguities that arise due to the operator precedence. The ambiguity occurs because there are no explicit rules to define the precedence of the operators. We'll disambiguate the grammar by introducing precedence and associativity rules for the operators '+', '-', and '×'.

To achieve this, we can modify the grammar as follows:

- **Introduce precedence levels for each operator:** Let's use the following precedence levels: '+' and '-' have lower precedence than '×'.
- **Rewrite the grammar with the precedence:** We'll use non-terminals to represent different precedence levels.

Modified Grammar:

E -> E + T | E - T | T

T -> T * F | F

F -> (E) | V

V -> x | y | z

Explanation:

- We've introduced non-terminals E, T, and F to represent different precedence levels. E represents the highest precedence level, T the intermediate level, and F the lowest level.
- The production rules have been rearranged to reflect the new precedence rules:
  - '+' and '-' have the same precedence and left-associativity, so they are on the same level (E) and have higher precedence than '*' (T).
  - '*' has lower precedence than '+' and '-', so it is on the T level.

This modified grammar ensures that expressions are parsed unambiguously with respect to operator precedence. Now, each expression will have a unique parse tree and derivation.

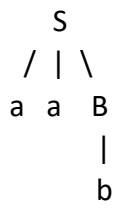
$$S \rightarrow AB \mid aaB$$
$$A \rightarrow a \mid Aa$$
$$B \rightarrow b$$

Derive the string **aab** from the grammar. Convert the grammar to unambiguous

Ans: Derivation 1:

S => aaB [Apply S → aaB]
=> aab [Apply B → b]

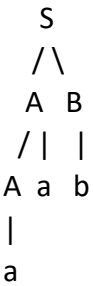
Parse Tree 1:



Derivation 2:

S => AB [Apply S → AB]
=> AaB [Apply A → Aa]
=> aaB [Apply A → a]
=> aab [Apply B → b]

Parse Tree 2:



As you can see, there are two different derivations and parse trees for the string "aab" using the given ambiguous grammar. This ambiguity arises because the grammar allows two different

ways to derive the same string, leading to multiple parse trees. This confirms the ambiguity of the grammar.

- What is causing the ambiguity?

The ambiguity in the grammar is caused by the production rule "S → aaB." This production allows the non-terminal "S" to derive the string "aab" in two different ways: either by directly using "S → aaB" or by using "S → AB" with "A → aa" and "B → b."

- Where are we allowed to make multiple choices?

The ambiguity arises when we have more than one production rule that can be applied to derive the same string. In this grammar, the non-terminal "S" can derive "aab" using both "S → aaB" and "S → AB."

- Can we remove the multiple choice option without removing the power of generating the same strings?

Yes, we can remove the ambiguity while still generating the same set of strings (ab, aab, aaab……..)

So removing S -> aaB the unambiguous grammar is given below:

S → AB
A → a | Aa
B → b