Unambiguous Grammar
A grammar can be unambiguous if the grammar does not contain ambiguity that means if it does not contain more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string.

It is important to note that there are **no direct algorithms** to find whether grammar is ambiguous or not. We need to build the parse tree for a given input string that belongs to the language produced by the grammar and then decide whether the grammar is ambiguous or unambiguous based on the number of parse trees obtained as discussed above.

Notes –
- There exists no general algorithm to remove the ambiguity from grammar, and it is not always possible to convert an ambiguous grammar into an unambiguous grammar.
- To check grammar ambiguity, we try finding a string that has more than one parse tree.
- If any such string exists, then the grammar is ambiguous otherwise not.
- The string has to be chosen carefully because there may be some strings available in the language produced by the unambiguous grammar which has only one parse tree.

Removal of Ambiguity:
We can remove ambiguity solely on the basis of the following two properties:

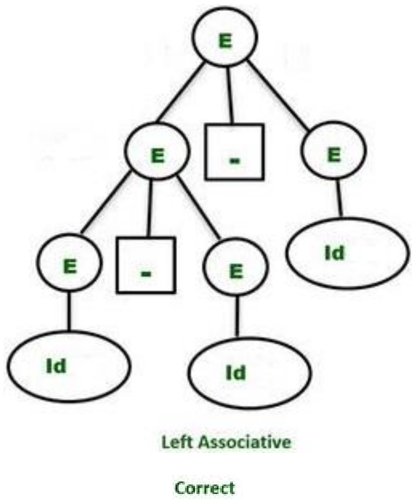
1. Precedence Constraints
If different operators are used, we will consider the precedence of the operators. The three important characteristics are:
1. The level at which the production is present denotes the priority of the operator used.
2. The production at **higher levels** will have **operators with less priority**. In the parse tree, the nodes which are at top levels or close to the root node will contain the lower priority operators.
3. The production at **lower levels** will have operators with **higher priority**. In the parse tree, the nodes which are at lower levels or close to the leaf nodes will contain the higher priority operators.

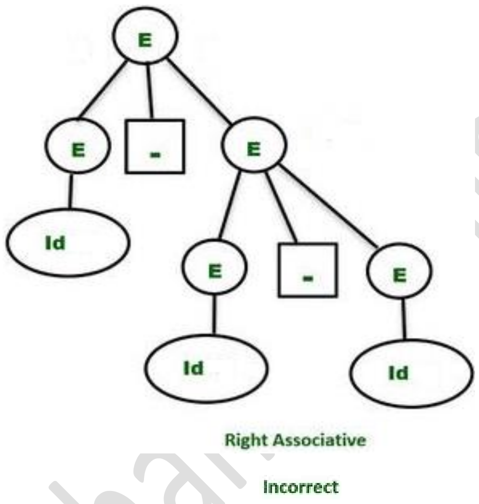
2. Associativity Constraints
If the same precedence operators are in production, then we will have to consider the associativity.
1. If the associativity is left to right, then we have to prompt a left recursion in the production. The parse tree will also be left recursive and grow on the left side.
   +, -, *, / are left associative operators.
   Left recursion means that the leftmost symbol on the right side is the same as the non-terminal on the left side. For example, $X \rightarrow Xa$.
2. If the associativity is right to left, then we have to prompt the right recursion in the productions. The parse tree will also be right recursive and grow on the right side.
   ^ is a right associative operator.
   Right recursion means that the rightmost symbol on the right side is the same as the non-terminal on the left side. For example, $X \rightarrow aX$

Example 1.
Consider the ambiguous grammar:
$E \rightarrow E - E \mid id$

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.

Solution
The language in the grammar will contain {id, id-id, id-id-id, ….}. Say, we want to derive the string "**id-id-id**". Since the same priority operators, we need to consider **associativity** which is left to right. The parse tree which grows on the left side of the root will be the correct parse tree in order to make the grammar unambiguous.



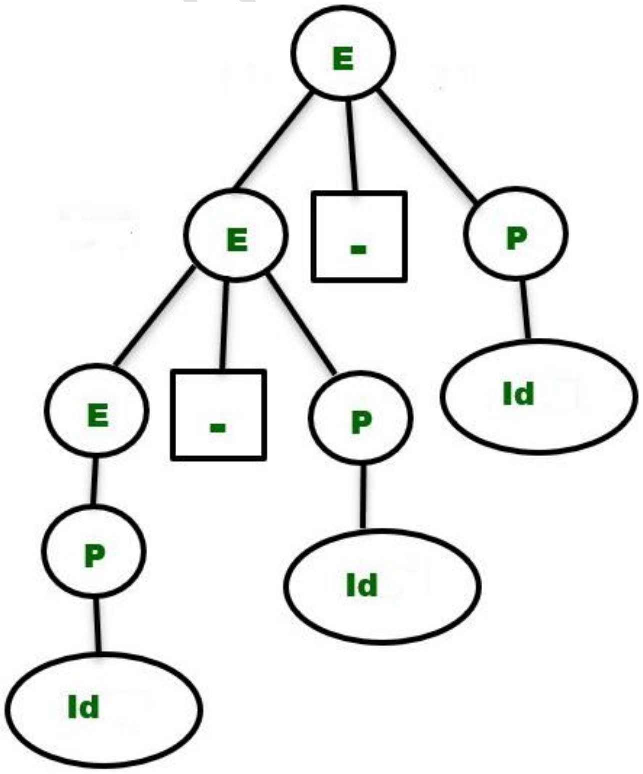


So, to make the above grammar unambiguous, simply make the grammar **Left Recursive** by replacing the left most non-terminal E in the right side of the production with another random variable, say **P.** The grammar becomes:

$E \rightarrow E - P \mid P$

$P \rightarrow id$

The above grammar is now unambiguous and will contain only one Parse Tree for the above expression as shown below:



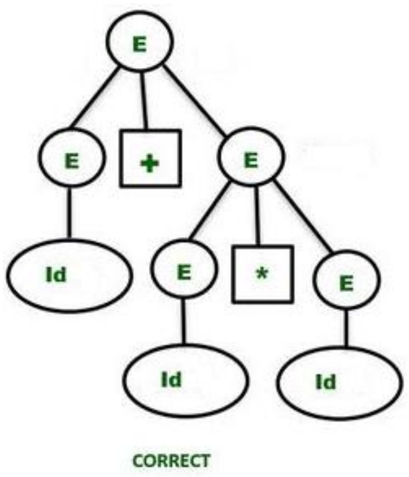
(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.

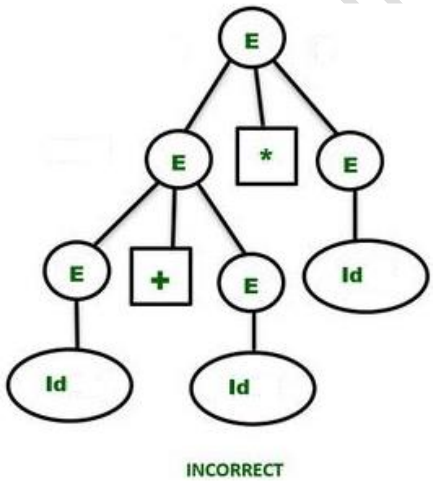
Similarly, the unambiguous grammar for the expression: "**id^id^id**" will be:

E → P ^ E | P   *// Right Recursive as ^ is right associative.*
P → id

Example 2.
Consider the grammar shown below, which has two different operators:
E → E+E | E*E | id

Solution
Clearly, the above grammar is ambiguous as we can draw two parse trees for the string "**id+id*id**" as shown below: (where "*" has more priority than "+").





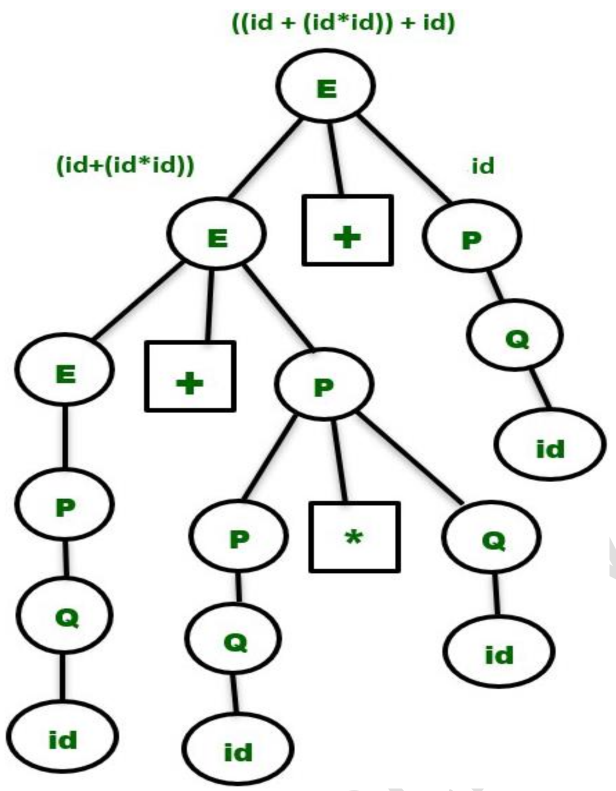
The "+" having the least priority has to be at the upper level and has to wait for the result produced by the "*" operator which is at the lower level. So, the first parse tree is the correct one and gives the same result as expected.

The unambiguous grammar will contain the productions having the highest priority operator **("*" in the example)** at the lower level and vice versa. The associativity of both the operators are **Left to Right**. So, the unambiguous grammar has to be **left recursive.** The grammar will be:

E → E + P | P   *// + is at higher level and left associative.*
P → P * Q | Q   *// * is at lower level and left associative.*
Q → id

E is used for doing addition operations and **P** is used to perform multiplication operations. They are independent and will maintain the precedence order in the parse tree.
The parse tree for the string "**id+id*id+id**" will be:

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.



Note: It is very important to note that while converting an ambiguous grammar to an unambiguous grammar, we shouldn't change the original language provided by the ambiguous grammar. So, the non-terminals in the ambiguous grammar have to be replaced with other variables in such a way that we get the same language as it was derived before and also maintain the precedence and associativity rule simultaneously. This is the reason we wrote the production **E → P** and **P → Q** and **Q → id** after replacing them in the above example, because the language contains the strings **{id, id+id}** as well.

Similarly, the unambiguous grammar for an expression having the operators **-, *,** ^ is:

E → E – P | P // Minus operator is at higher level due to least priority and left associative.
P → P * Q | Q // Multiplication operator has more priority than – and lesser than ^ and left associative.
Q → R ^ Q | R // Exponent operator is at lower level due to highest priority and right associative.
R → id

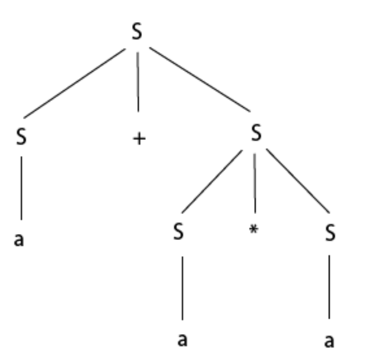
Note: Also, there are some ambiguous grammars which can't be converted into unambiguous grammars.

Example 3.
Check that the given grammar is ambiguous or not. Also, find an equivalent unambiguous grammar:

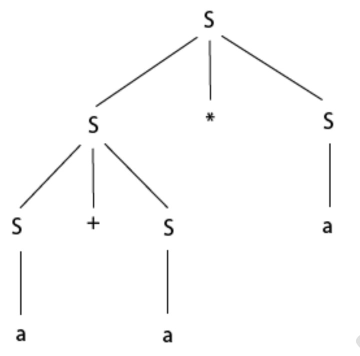
S → S+S | S*S | S^S | a

Solution
The given grammar is ambiguous because the derivation of string "**a+a*b**" can be represented by two different parse trees:

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.



Parse tree 1



Parse tree 2

Unambiguous grammar will be:

$S \rightarrow S + A \mid A$

$A \rightarrow A * B \mid B$

$B \rightarrow C \wedge B \mid C$

$C \rightarrow a$

HW. *Draw the parse tree of the above Unambiguous grammar for the string "**a+a*b**".*

Example 4.

Check that the given context free grammar is ambiguous or not:

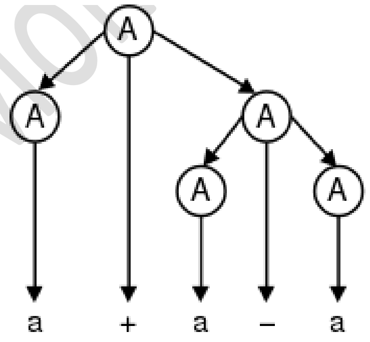
$A \rightarrow A+A \mid A-A \mid a$

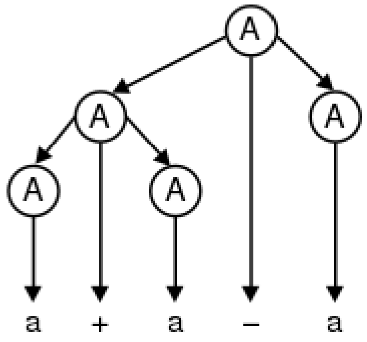
Solution

The given grammar is ambiguous since there are two leftmost derivations for the string "**a+a+a**":

A	⇒ A+A	A	⇒ A+A
	⇒ a+A		⇒ A+A+A (First A is replaced by A+A. Replacement of the second A would yield a similar derivation).
	⇒ a+A+A		⇒ a+A+A
	⇒ a+a+A		⇒ a+a+A
	⇒ a+a+a		⇒ a+a+a

As **another example**, the grammar is ambiguous since there are two parse trees for the string "**a+a−a**":





The language that it generates, however, is not inherently ambiguous; the following is a non-ambiguous grammar generating the same language:

$A \rightarrow A+a \mid A-a \mid a$

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.

Example 5.
Convert the following ambiguous grammar into unambiguous grammar:
$E \rightarrow E + E \mid E\ .\ E \mid E^* \mid a \mid b$
where ***** is kleen closure and **.** is concatenation.

Solution
To convert the given grammar into its corresponding unambiguous grammar, we implement the precedence and associativity constraints.
We have-
- Given grammar consists of the following operators:
  +, . , *
- Given grammar consists of the following operands:
  a, b

The priority order is-
  (a, b) > * > . > +
where-
- **.** operator is left associative
- **+** operator is left associative

Using the precedence and associativity rules, we write the corresponding unambiguous grammar as:

$E \rightarrow E + T \mid T$
$T \rightarrow T\ .\ F \mid F$
$F \rightarrow F^* \mid a \mid b$ OR: $F \rightarrow F^* \mid G$
$G \rightarrow a \mid b$

HW. *Suggest a string and then draw the parse tree of the above Unambiguous grammar.*

HW 1. *Consider the production shown below. Check that the given grammar is ambiguous or not:*
$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

HW 2. *Consider the production shown below. Check that the given grammar is ambiguous or not:*
$S \rightarrow AB$
$A \rightarrow Aa \mid a$
$B \rightarrow b$

HW 3. *Convert the following ambiguous grammar into unambiguous grammar:*
bexp → bexp or bexp | bexp and bexp | not bexp | T | F

- where bexp represents Boolean expression, T represents True and F represents False.
- We have-
  - Given grammar consists of the following operators:
    or, **and**, **not**
  - Given grammar consists of the following operands:
    T, **F**
- The priority order is-
  - (**T**, **F**) > **not** > **and** > **or**

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.

HW 4. *Consider a grammar G is given as follows:*

S → AB | aaB
A → a | Aa
B → b

Determine whether the grammar G is ambiguous or not. If G is ambiguous, construct an unambiguous grammar equivalent to G.

HW 5. *Show that the given grammar is ambiguous. Also, find an equivalent unambiguous grammar:*

S → ABA
A → aA | ε
B → bB | ε

Difference between Ambiguous and Unambiguous Grammar

1. Ambiguous Grammar:

A context-free grammar is called ambiguous grammar if there exists more than one derivation tree or parse tree.

2. Unambiguous Grammar:

A context-free grammar is called unambiguous grammar if there exists one and only one derivation tree or parse tree.

S. NO	Ambiguous Grammar	Unambiguous Grammar
1.	In ambiguous grammar, the leftmost and rightmost derivations are not same.	In unambiguous grammar, the leftmost and rightmost derivations are same.
2.	Amount of non-terminals in ambiguous grammar is less than in unambiguous grammar.	Amount of non-terminals in unambiguous grammar is more than in ambiguous grammar.
3.	Length of the parse tree in ambiguous grammar is comparatively short.	Length of the parse tree in unambiguous grammar is comparatively large.
4.	Speed of derivation of a tree in ambiguous grammar is faster than that of unambiguous grammar.	Speed of derivation of a tree in unambiguous grammar is slower than that of ambiguous grammar.
5.	Ambiguous grammar generates more than one parse tree.	Unambiguous grammar generates only one parse tree.
6.	Ambiguous grammar contains ambiguity.	Unambiguous grammar does not contain any ambiguity.

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.