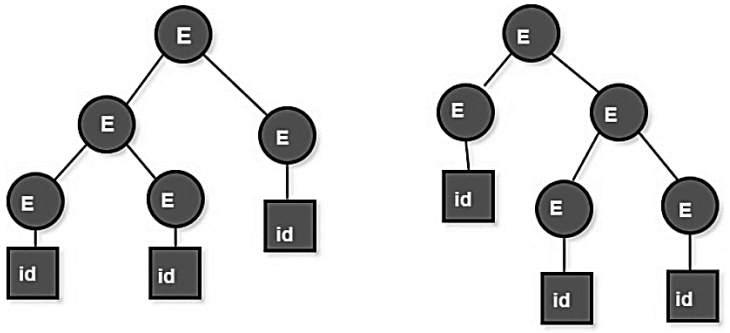
Ambiguous grammar: A CFG is said to be ambiguous if there exists more than one derivation tree for the given input string i.e., more than one **L**eft**M**ost **D**erivation **T**ree (LMDT) or **R**ight**M**ost **D**erivation **T**ree (RMDT).

Grammar with strings that have two or more distinct parse trees, are called ambiguous. Assigning a parse tree to a given string in the language is called parsing the string, it is an important first step towards understanding the structure of the string. Ambiguous grammars are of no help in parsing, since they assign no unique parse tree to each string in the language.

Example 1: Let us consider this grammar: **E -> E+E|id** We can create two parse trees from this grammar to obtain a string **id+id+id.** The following are the two parse trees generated by left-most derivation:



Both the above parse trees are derived from the same grammar rules but both parse trees are different. Hence the grammar is ambiguous.

Example 2: Let us now consider the following grammar:

Set of alphabets ∑ = {0,…,9, +, *, (,)}

E -> I

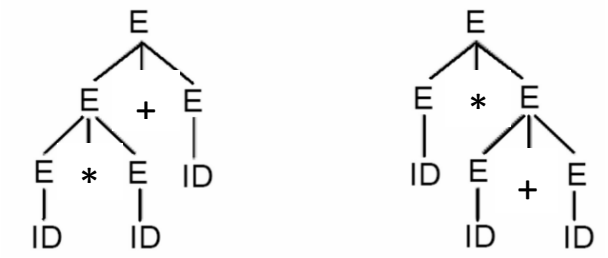
E -> E + E

E -> E * E

E -> (E)

I -> ε | 0 | 1 | … | 9

From the above grammar String 3*2+5 can be derived in 2 ways:



I) First leftmost derivation

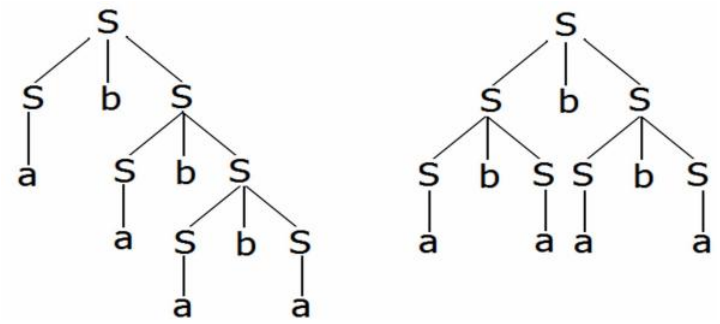
E=>E*E
=>I*E
=>3*E+E
=>3*I+E
=>3*2+E
=>3*2+I
=>3*2+5

II) Second leftmost derivation

E=>E+E
=>E*E+E
=>I*E+E
=>3*E+E
=>3*I+E
=>3*2+I
=>3*2+5

Example 3: If G is the grammar: S →SbS|a

To prove that G is ambiguous, there are need to find a w ∈ L(G), which is ambiguous. Consider the word abababa.



Following are some examples of ambiguous grammar:

- S→ aS |Sa| Є
- E→ E +E | E*E| id
- A→ AA | (A) | a
- S → SS|AB , A → Aa|a , B → Bb|b

Whereas following grammars are unambiguous:

- S → (L) | a, L → LS | S
- S→ AA , A → aA , A → b

Exercise: Consider the grammar P={S→aS | aSbS | ϵ } , construct the string aab with rightmost and leftmost derivation and draw the parse tree, explaining if this grammar ambiguous?

Simplifying Context-Free Grammars:

In order to construct good and efficient parsing algorithms, consider transformations of grammars, essentially simplifications of the form of productions, which lead to certain normal forms for grammars. These normal forms are better to deal with, and they allow more efficient parsing algorithms.

Types of redundant productions and the procedure of removing them are mentioned below:

1. Useless productions
2. Substitution Rule
3. Removing λ-productions
4. Removing unit productions

1. Useless productions:

The productions that can never take part in derivation of any string, are called useless productions. Similarly, a variable that can never take part in derivation of any string is called a useless variable.

Example 1:

$S \rightarrow abS \mid abA \mid abB$
$A \rightarrow cd$
$B \rightarrow aB$
$C \rightarrow dc$

Note that the concept of 'useless variable' includes

1: The case that the variable does not lead to a terminal string.
2: The case that the variable cannot be reached from the start symbol.

In the example above, production $C \rightarrow dc$ is useless because the variable 'C' will never occur in derivation of any string. The other productions are written in such a way that variable 'C' can never reached from the starting variable 'S'.

Production $B \rightarrow aB$ is also useless because there is no way it will ever terminate. If it never terminates, then it can never produce a string. Hence the production can never take part in any derivation.

The elimination of useless variables and productions from a grammar (or the selection of those variables and productions, which are useful proceeds in two phases:

A. Determine and select those variables, which can lead to a terminal string, and subsequently determine and select the related productions.

B. Determine and select those variables, which can be reached from the start symbol, and select related productions.

In the example above, to remove useless productions, we first find all the variables which will never lead to a terminal string such as variable 'B'. We then remove all the productions in which variable 'B' occurs.
So, the modified grammar becomes:
S → abS | abA
A→ cd
C → dc

We then try to identify all the variables that can never be reached from the starting variable such as variable 'C'. We then remove all the productions in which variable 'C' occurs. The grammar below is now free of useless productions

S→ abS | abA
A → cd

Example 2:

S→ aSb | λ |A
A→aA
Cannot generate a terminal string, so variable A is useless.

Example 3:

S→A
A→aA| λ
B→bA
Cannot be reached from S, so variable B is useless.

Example 4: Eliminate useless symbols from the grammar with productions:

S →AB | CA
B →BC |AB
A →a
C →AB | b
Step 1: Eliminate non-generating symbols, so the variables and productions will be:
V1 = {A, C, S}
P1 = {S →CA, A →a, C →b}
Step 2: Eliminate symbols that are non-reachable.
All Variables are reachable. So, the final variables and productions are same V1 and P1.
V2 = {A, C, S}
P2 = {S →CA, A →a, C →b}

Exercises: Eliminate useless symbols from the following grammars:

1. P= {S →aAa, A →Sb | bCC, C →abb, E →aC}

2. P= {S →aBa | BC, A →aC | BCC, C →a, B →bcc, D →E, E →d}

3. P= {S →aAa, A →bBB, B →ab, C →aB}

4. P= {S →aS | AB, A →bA, B →AA}

2- Substitution Rule:

If a grammar contains a production $A \rightarrow x_1 B x_2$ with A, B ∈ V and A ≠ B and $B \rightarrow y_1 \mid y_2 \mid ... \mid y_n$ (alternatives) then construct a new grammar G' with a modified set of productions P', in which replace the rules for A and B above with one set of rules: $A \rightarrow x_1 y_1 x_2 \mid x_1 y_2 x_2 \mid ... \mid x_1 y_n x_2$ G and G' are equivalent, i.e. the generate the same language.

Example 5:

A → a | aaA | abBc

B → abbA | b

This grammar is equivalent to:

A→ a | aaA | ababbAc | abbc

3. Removing λ-productions

The productions of type 'A → λ' are called λ productions (also called *lambda* productions and *null* productions). These productions can only be removed from those grammars that do not generate λ (an empty string). It is possible for a grammar to contain null productions and yet not produce an empty string.

To remove null productions:

1. we first have to find all the nullable variables. A variable 'A' is called nullable if λ can be derived from 'A'. For all the productions of type 'A → λ', 'A' is a nullable variable. For all the productions of type 'B → A1A2…An ', where all 'Ai's are nullable variables, 'B' is also a nullable variable.

Example 1:
S→ABaC
A→BC
B→b| λ
C→D| λ
D→d
Nullable set VN={A, B, C}

2. After finding all the nullable variables, we can now start to construct the null production free grammar. For all the productions in the original grammar, we add the original production as well as all the combinations of the production that can be formed by replacing the nullable variables in the production by λ. If all the variables on the RHS of the production are nullable, then we do not add 'A → λ' to the new grammar. In another words, for two nullable variables on the RHS, remove one, remove the other, remove both, remove none. An example will make the point clear.

Example 2:

Consider the grammar:

S → ABCd (1)
A → BC (2)
B → bB | λ (3)
C → cC | λ (4)

- First find all the nullable variables. Variables 'B' and 'C' are clearly nullable because they contain 'λ' on the RHS of their production. Variable 'A' is also nullable because in (2), both variables on the RHS are also nullable. So, variables 'A', 'B' and 'C' are nullable variables.
- Create the new grammar. We start with the first production. Add the first production as it is. Then we create all the possible combinations that can be formed by replacing the nullable variables with λ. Therefore, line (1) now becomes 'S → ABCd | ABd | ACd | BCd | Ad | Bd |Cd | d'. We apply the same rule to line (2) but we do not add 'A → λ' even though it is a possible combination. We remove all the productions of type 'V → λ'. The new grammar now becomes:

S → ABCd | ABd | ACd | BCd | Ad | Bd |Cd | d
A → BC | B | C
B → bB | b
C → cC | c

Example 3:

Find out the grammar without Ɛ- Productions G = ({S, A, B, D}, {a}, {S →aS | AB, A → λ, B→ λ, D →b}, S)

Nullable variables = {S, A, B}

New Set of productions:

S → aS | a
S → AB | A | B
D →b

G1= ({S, B, D}, {a}, {S →aS | a | AB | A | B, D →b}, S)

Exercises: Eliminate λ - productions from the grammar

1. S →a |Xb | aYa, X →Y| λ, Y →b | X
2. S →Xa, X →aX | bX | λ
3. S →XY, X →Zb, Y →bW, Z →AB, W →Z, A →aA | bB | λ, B →Ba | Bb| λ
4. S →ASB | λ, A →aAS | a, B →SbS | A| bb

Example 4:

Eliminate Ɛ - productions and useless symbols from the grammar

S →a |aA|B|C

A →aB| λ
B →aA
C →aCD
D →ddd
Step 1: Eliminate Ɛ - productions
Nullable ={A}
P1={S →a |aA | B | C, A →aB, B →aA|a, C →aCD, D →ddd}
Step 2: Eliminate useless productions
a: Eliminate non-generating symbols, Generating ={D, B, A, S}
P2={S →a | aA| B, A →aB, B →aA|a, D →ddd}
b: Eliminate non -reachable symbols, D is non-reachable, eliminating.
P3= {S →a |aA|B, A →aB, B →aA|a}

4. Unit productions:

The productions of type 'A → B' are called unit productions. To create a unit production free grammar 'Guf' from the original grammar 'G' , we follow the procedure mentioned below.
First add all the non-unit productions of 'G' in 'Guf'. Then for each variable 'A' in grammar 'G' , find all the variables 'B' such that 'A *=> B'. Now , for all variables like 'A ' and 'B', add 'A → x1 | x2 | …xn' to 'Guf' where 'B → x1 | x2 | …xn ' is in 'Guf' . None of the x1 , x2 … xn are single variables because we only added non-unit productions in 'Guf'. Hence the resultant grammar is unit production free.
Example 5:
S → Aa | B
A → b | B
B → A | a
Lets add all the non-unit productions of 'G' in 'Guf'. 'Guf' now becomes:
S → Aa
A → b
B → a
Now we find all the variables that satisfy 'X *=> Z'. These are 'S*=>B', 'A *=> B' and 'B *=> A'. For 'A *=> B' , we add 'A → a' because 'B ->a' exists in 'Guf'. 'Guf' now becomes:
S → Aa
A → b | a
B → a
For 'B *=> A' , we add 'B → b' because 'A → b' exists in 'Guf'. The new grammar now becomes
S → Aa
A → b | a
B → a | b
We follow the same step for 'S*=>B' and finally get the following grammar
S → Aa | b | a
A → b | a
B → a | b

Now remove B → a|b , since it doesnt occur in the production 'S', then the following grammar becomes,

S→Aa|b|a

A→ b|a

Note: To remove all kinds of productions mentioned above, first remove the null productions, then the unit productions and finally, remove the useless productions. Following this order is very important to get the correct result.

Example 6: Simplify this grammar:

S→Aa|B

B→A|bb

A→a|bc|B

Remove unit productions (S→B, B→A, A→B)

S→Aa

B→bb

A→a|bc

Add:

S→bb|a|bc

A→bb

B→a|bc

Finally, we get:

S→a|bc|bb|Aa

A→a|bc|bb

B→a|bc|bb