CONTEXT-FREE GRAMMAR (CFG)

CFG stands for context-free grammar. It is is a formal grammar which is used to generate all possible patterns of strings in a given formal language. Context-free grammar G can be defined by four tuples as:

G = (V, T, P, S)

Where,

G is the grammar, which consists of a set of the production rule. It is used to generate the string of a language.

T is the final set of a terminal symbol. It is denoted by lower case letters.

V is the final set of a non-terminal symbol. It is denoted by capital letters.

P is a set of production rules, which is used for replacing non-terminals symbols(on the left side of the production) in a string with other terminal or non-terminal symbols(on the right side of the production).

S is the start symbol which is used to derive the string. We can derive the string by repeatedly replacing a non-terminal by the right-hand side of the production until all non-terminal have been replaced by terminal symbols.

Example :

Construct the CFG for the language having any number of a's over the set ∑= {a}.

Solution:

As we know the regular expression for the above language is

1. r.e. = a*

Production rule for the Regular expression is as follows:

1. S → aS rule 1
2. S → ε rule 2

Now if we want to derive a string "aaaaaa", we can start with start symbols.

1. S
2. aS
3. aaS rule 1
4. aaaS rule 1
5. aaaaS rule 1
6. aaaaaS rule 1

7. aaaaaaS rule 1
8. aaaaaaε rule 2
9. aaaaaa

The r.e. = a* can generate a set of string {ε, a, aa, aaa,.....}. We can have a null string because S is a start symbol and rule 2 gives S → ε.

Example :

Construct a CFG for the regular expression (0+1)*

Solution:

The CFG can be given by,

1. Production rule (P):
2. S → 0S | 1S
3. S → ε

The rules are in the combination of 0's and 1's with the start symbol. Since (0+1)* indicates {ε, 0, 1, 01, 10, 00, 11, ...}. In this set, ε is a string, so in the rule, we can set the rule S → ε.

Example :

Construct a CFG for a language L = {wcwR | where w € (a, b)*}.

Solution:

The string that can be generated for a given language is {aacaa, bcb, abcba, bacab, abbcbba, ...}

The grammar could be:

1. S → aSa rule 1
2. S → bSb rule 2
3. S → c rule 3

Now if we want to derive a string "abbcbba", we can start with start symbols.

1. S → aSa
2. S → abSba from rule 2
3. S → abbSbba from rule 2
4. S → abbcbba from rule 3

Thus any of this kind of string can be derived from the given production rules.

Example 4:

Construct a CFG for the language L = $a^n b^{2n}$ where n>=1.

Solution:

The string that can be generated for a given language is {abb, aabbbb, aaabbbbbb....}.

The grammar could be:

1. S → aSbb | abb

Now if we want to derive a string "aabbbb", we can start with start symbols.

1. S → aSbb
2. S → aabbbb

Derivation

Derivation is a sequence of production rules. It is used to get the input string through these production rules. During parsing, we have to take two decisions. These are as follows:

- We have to decide the non-terminal which is to be replaced.
- We have to decide the production rule by which the non-terminal will be replaced.

We have two options to decide which non-terminal to be placed with production rule.

1. Leftmost Derivation:

In the leftmost derivation, the input is scanned and replaced with the production rule from left to right. So in leftmost derivation, we read the input string from left to right.

Example:

Production rules:

1. E = E + E
2. E = E - E
3. E = a | b

Input

1. a - b + a

The leftmost derivation is:

1. E = E + E
2. E = E - E + E
3. E = a - E + E
4. E = a - b + E
5. E = a - b + a

2. Rightmost Derivation:

In rightmost derivation, the input is scanned and replaced with the production rule from right to left. So in rightmost derivation, we read the input string from right to left.

Example

Production rules:

1. E = E + E
2. E = E - E
3. E = a | b

Input

1. a - b + a

The rightmost derivation is:

1. E = E - E
2. E = E - E + E
3. E = E - E + a
4. E = E - b + a
5. E = a - b + a

When we use the leftmost derivation or rightmost derivation, we may get the same string. This type of derivation does not affect on getting of a string.

Examples of Derivation:

Example :

Derive the string "abb" for leftmost derivation and rightmost derivation using a CFG given by,

1. S → AB | ε
2. A → aB
3. B → Sb

Solution:

Leftmost derivation:

S

AB

$\boxed{aB}$ B

a $\boxed{Sb}$ B

A $\boxed{\varepsilon}$ bB

ab $\boxed{Sb}$

ab $\boxed{\varepsilon}$ b

abb

Rightmost derivation:

S

AB

A $\boxed{Sb}$

A $\boxed{\varepsilon}$ b

$\boxed{aB}$ b

a $\boxed{Sb}$ b

a $\boxed{\varepsilon}$ bb

abb

Example :

Derive the string "aabbabba" for leftmost derivation and rightmost derivation using a CFG given by,

1. S → aB | bA
2. S → a | aS | bAA
3. S → b | aS | aBB

Solution:

Leftmost derivation:

1. S
2. aB S → aB
3. aaBB B → aBB

4. aabB $B \rightarrow b$
5. aabbS $B \rightarrow bS$
6. aabbaB $S \rightarrow aB$
7. aabbabS $B \rightarrow bS$
8. aabbabbA $S \rightarrow bA$
9. aabbabba $A \rightarrow a$

Rightmost derivation:

1. S
2. aB $S \rightarrow aB$
3. aaBB $B \rightarrow aBB$
4. aaBbS $B \rightarrow bS$
5. aaBbbA $S \rightarrow bA$
6. aaBbba $A \rightarrow a$
7. aabSbba $B \rightarrow bS$
8. aabbAbba $S \rightarrow bA$
9. aabbabba $A \rightarrow a$

Example :

Derive the string "00101" for leftmost derivation and rightmost derivation using a CFG given by,

1. $S \rightarrow A1B$
2. $A \rightarrow 0A \mid \varepsilon$
3. $B \rightarrow 0B \mid 1B \mid \varepsilon$

Solution:

Leftmost derivation:

1. S
2. A1B
3. 0A1B
4. 00A1B
5. 001B
6. 0010B
7. 00101B
8. 00101

Rightmost derivation:

1. S

2. A1B
3. A10B
4. A101B
5. A101
6. 0A101
7. 00A101
8. 00101