Context-Free Grammar (CFG)

CFG stands for context-free grammar. It 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,

**S** is the start symbol

### **Example 1:**

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.

 S

aS

aaS          rule 1

aaaS         rule 1

aaaaS        rule 1

aaaaaS       rule 1

aaaaaaS      rule 1

aaaaaaε      rule 2

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 2:**

Construct a CFG for the language L = anb2n where n>=1.

**Solution:**

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

The grammar could be:

S → aSbb | abb

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

S → aSbb

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:**

E = E + E

E = E - E

E = a | b

**Input**

a - b + a

**The leftmost derivation is:**

E = E + E

E = E - E + E

E = a - E + E

E = a - b + E

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:**

E = E + E

E = E - E

E = a | b

**Input**

a - b + a

**The rightmost derivation is:**

E = E - E

E = E - E + E

E = E - E + a

E = E - b + a

E = a - b + a

### **Example 2:**

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

S → aB | bA

S → a | aS | bAA

S → b | aS | aBB

**Leftmost derivation:**

S

aB            S → aB

aaBB          B → aBB

aabB          B → b

aabbS         B → bS

aabbaB        S → aB

aabbabS       B → bS

aabbabbA      S → bA

aabbabba      A → a

**Rightmost derivation:**

S

aB            S → aB

aaBB          B → aBB

aaBbS         B → bS

aaBbbA        S → bA

aaBbba        A → a

aabSbba       B → bS

aabbAbba      S → bA

aabbabba      A → a

# Derivation Tree

Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from a given set of production rules. The derivation tree is also called a parse tree.

A parse tree contains the following properties:

1. The root node is always a node indicating start symbols.
2. The derivation is read from left to right.
3. The leaf node is always terminal nodes.
4. The interior nodes are always the non-terminal nodes.

### **Example 1:**

**Production rules:**

E = E + E

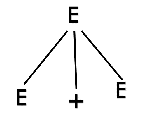
E = E \* E

E = a | b | c

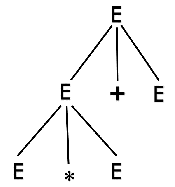
**Input**

a \* b + c

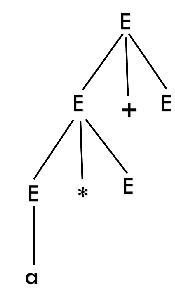
**Step 1:**



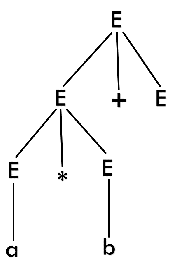
**Step 2:**



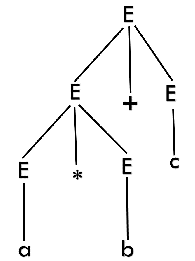
**Step 3:**



**Step 4:**



**Step 5:**



# Ambiguity in Grammar

A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string. If the grammar is not ambiguous, then it is called unambiguous.

### **Example:**

Check whether the given grammar G is ambiguous or not.

1. E → E + E
2. E → E - E
3. E → id

**Solution:**

From the above grammar String "id + id - id" can be derived in 2 ways:

**First Leftmost derivation**

E → E + E

   → id + E

   → id + E - E

   → id + id - E

   → id + id- id

**Second Leftmost derivation**

E → E - E

   → E + E - E

   → id + E - E

   → id + id - E

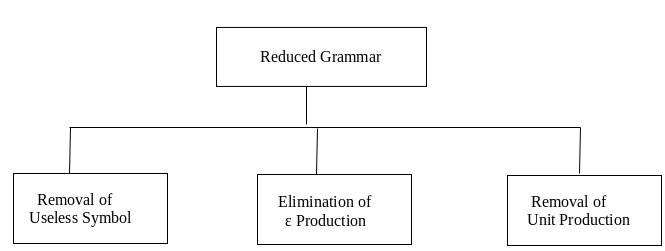
   → id + id - id

Since there are two leftmost derivations for a single string "id + id - id", the grammar G is ambiguous.

Simplification of CFG

Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below:

1. Each variable (i.e. non-terminal) and each terminal of G appears in the derivation of some word in L.
2. There should not be any production as X → Y where X and Y are non-terminal.
3. If ε is not in the language L then there need not to be the production X → ε.



1. Eliminate Useless Symbols

Step 1: Eliminate productions which contains non generating symbols

Step 2: Eliminate productions which contains non reachable symbols

Ex: consider a CFG

S→AB|0 A→1

▪B is non generating.so eliminate B ie.AB

▪S→0,A→1

▪A is non reachable.

Remove A.

final grammar becomes ▪S→0

1. Eliminate Unit Productions

▪Productions of the form A→B is called Unit production

▪Step 1: Select a unit production A→B such that there exist a production B→α, where α is a terminal

▪Step 2: For every non unit production B→α, add A→α to the grammar.

Remove A→B from the grammar

▪Ex: ▪A→B B→1

▪Replace by A→1 and remove A→B

3.Removal of ε Productions and nullable non terminal

* If a CFG contains productions of the form A→ ε, then this production is used to erase A, where A is a non-terminal known as nullable non terminal
* Eliminating productions of the form A→ ε, are called removal of ε production
* If A→ε is a production to be eliminated, whose right side contains A, replace each occurrence of A in each of these productions to obtain non ε productions. Add these new productions to the grammar
* S→0A A→1| ε
* A→ ε is the ε production. So, replace ε in place of A, at the right side of productions and add the resultant productions to the grammar
* S→0A become
* S→0 Add this to the grammar and remove A→ ε
* Final grammar is S→0A, S→0, A→1

*Examples discussed in the class.*

**Normal Forms**

Normalization is performed in order to standardize the grammar.

▶ By reducing the grammar, the grammar gets minimized but does not gets standardized.

▶ This is because the RHS of productions have no specific format.

▶ In order to standardize the grammar, normalization is performed using normal forms.

▶ The most frequently used normal forms are-

▶**Chomsky Normal Form (CNF)**

▶ **Greibach Normal Form (GNF)**

**Chomsky Normal Form (CNF)**

A Grammar is said to be in CNF, if every production rule of the grammar is of the form

**A → BC or A → a**

where A, B, C are non-terminals and a is a terminal.

To be in CNF, all the productions must derive either wo non-terminals or a single terminal.

CNF restricts the number of symbols on the right side of a production to be two.

The two symbols must be non-terminals or a single terminal.

**Steps for converting to CNF**

* Remove ɛ or unit production from the grammar
* Eliminate terminals on RHS,ie. If the grammar contains productions of the form S->bA then replace it by S->CbA , Cb->b
* Limit the number of variables on the RHS by 2

*\*Check the examples discussed in class*

**Greibach Normal Form (GNF)**

## A grammar is said to be in GNF, if every production is in the form

**A**->**aα**

## A is a non-terminal and a is a terminal and α is a string of non-terminals

*S->aA GNF*

*S->a GNF*

*S->AA not GNF*

*S->Aa not GNF*

## **Steps for converting CFG into GNF**

**Step 1:** Convert the grammar into CNF.

**Step 2:** If the grammar exists left recursion, eliminate it.

**Step 3:** In the grammar, convert the given production rule into GNF form.

*\*Check the examples discussed in class*