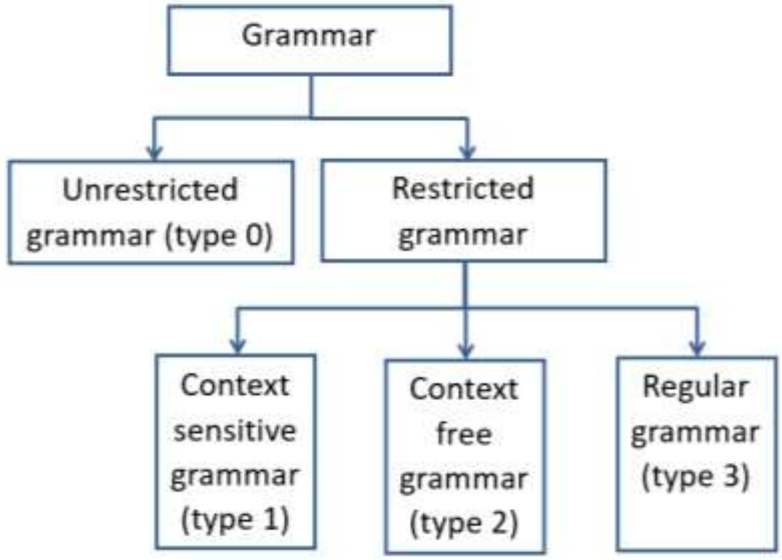
UNIT-III: Introduction

Table of Content

Chomsky hierarchy (Classification of grammar)



Type 0 grammar (Phrase Structure Grammar)

- Their productions are of the form:

$$\alpha \rightarrow \beta$$

- where both α and β can be strings of terminal and nonterminal symbols.
- Example: S → ACaB

  Bc → acB

  CB → DB

  aD → Db

Type 1 grammar (Context Sensitive Grammar)

- Their productions are of the form:

$$\alpha A\beta \rightarrow \alpha\pi\beta$$

- where **A** is non terminal and $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\pi}$ are strings of terminals and non terminals.
- The strings $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ may be empty, but $\boldsymbol{\pi}$ must be non-empty.
- Here, a string π can be replaced by $'A'$ (or vice versa) only when it is enclosed by the strings α and β in a sentential form.
- Example: AB → AbBc

  A → bcA

  B → b

Type 2 grammar (Context Free Grammar)

- Their productions are of the form:

$$A \rightarrow \alpha$$

- Where $\boldsymbol{A}$ is non terminal and $\boldsymbol{\alpha}$ is string of terminals and non terminals.
- Example: S → Xa

  X → a

  X → aX

  X → abc

Type 3 grammar (Linear or Regular grammar)

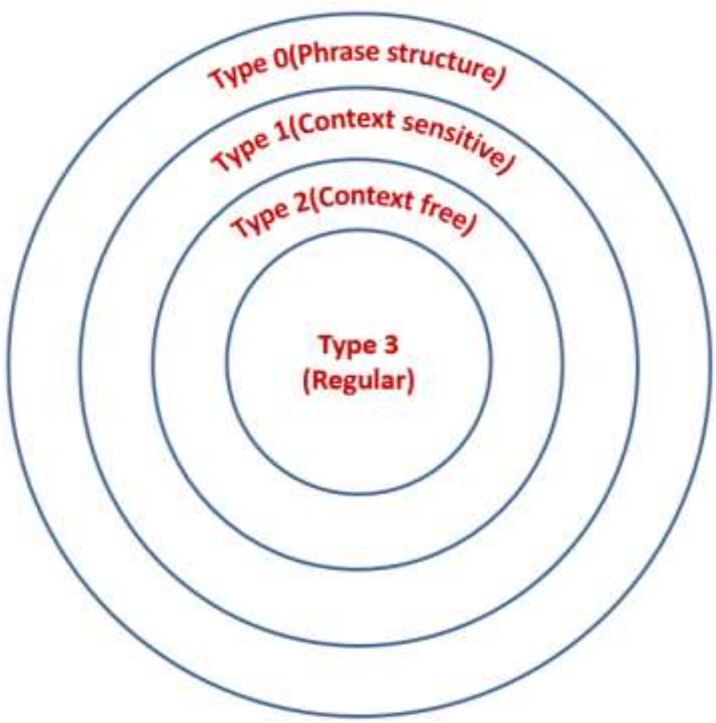
- Their productions are of the form:

$$A \rightarrow tB \mid t \quad \text{or} \quad A \rightarrow Bt \mid t$$

- Where A, B are non terminals and t is terminal.
- Example: X → a | aY

  Y → b

Hierarchy of grammar



Context Free Grammar

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

- A context free grammar (CFG) is a 4-tuple $G = (V, \Sigma, S, P)$ where,

  V is finite set of non terminals,

  Σ is disjoint finite set of terminals,

  S is an element of V and it's a start symbol,

  P is a finite set of productions of the form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$.

CFG Examples

- **Write CFG for either a or b**

  S→a | b

- **Write CFG for a⁺**

  S→ aS | a

- **Write CFG for a***

  S→ aS | ^

- **Write CFG for (ab)***

  S→abS | ^

- **Write CFG for any string of a and b**

  S→ aS | bS | a | b

CFG Examples

- **Write CFG for ab***

  S→aX

  X→^| bX

- **Write CFG for a*b***

  S→XY

  X→aX|^

  Y→bY|^

- **Write CFG for (a+b)***

  S→aS | bS | ^

- **Write CFG for a(a+b)***

  S→aX

  X→aX | bX | ^

CFG Examples

- **Write CFG for a* | b***

  S→A | B

  A→^| aA

  B→^ |bB

- **Write CFG for (011+1)*(01)***

  S→AB

  A→011A | 1A | ^

  B→01B | ^

- **Write CFG for balanced parenthesis**

  S→ [] | {} | [s] | {s} | ^

CFG Examples

- **Write CFG which contains at least three times 1.**

  S→A1A1A1A

  A→0A | 1A | ^

- **Write CFG that must start and end with same symbol.**

  S→0A0 | 1A1

  A→0A | 1A | ^

- **The language of even & odd length palindrome string over {a,b}**

  S→aSa|bSb|a|b|^

- **No. of a and no. of b are same**

  S→aSb|bSa|^

- **The language of {a, b} ends in a**

  S→aS | bS |a

CFG Examples

- **Write CFG for regular expression (a+b)*a(a+b)*a(a+b)***

  S→XaXaX

  X→aX|bX|^

- **Write CFG for number of 0's and 1's are same ($n_0(x)=n_1(x)$)**

  S→0S1 | 1S0 | ^

- **Write CFG for L={$a^ib^jc^k$ | i=j or j=k}**

For i=j	for j=k
S→AB	S→CD
A→aAb | ab	C→aC | a
B→cB | c	D→bDc | bc

CFG Examples

- **Write CFG for L={ $a^i b^j c^k$ | j>i+k}**

  S→ABC

  A→aAb |^

  B→bB | b

  C→bCc |^

- **Write CFG for L={ $0^i 1^j 0^k$ | j>i+k}**

  S→ABC

  A→0A1 |^

  B→1B | 1

  C→1C0 |^

- **Write CFG for the language of Algebraic expressions**

  S→S+S | S*S | S-S | S/S | (S) | a

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

Derivation

- Derivation is used to find whether the string belongs to a given grammar or not.
- There are two types of derivation:
  1. Leftmost derivation
  2. Rightmost derivation

Leftmost derivation

- A derivation of a string W in a grammar G is a left most derivation if at every step the left most non terminal is replaced.
- Grammar: S→S+S | S-S | S*S | S/S | a   Output string: a*a-a

S

→**S**-**S**

→**S*S**-S

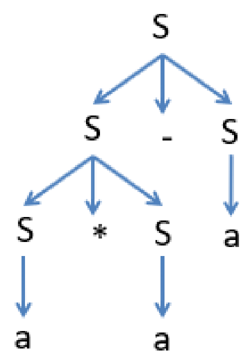
→**a***S-S

→a***a**-S

→a*a-**a**

Leftmost Derivation

Parse tree represents the structure of derivation



Parse tree

Rightmost derivation

- A derivation of a string W in a grammar G is a right most derivation if at every step the right most non terminal is replaced.
- It is all called canonical derivation.
- Grammar: S→S+S | S-S | S*S | S/S | a   Output string: a*a-a

S

→**S*S**

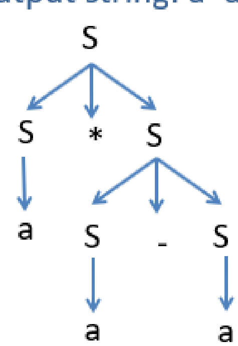
→S***S-S**

→S*S-**a**

→S***a**-a

→**a***a-a

Rightmost Derivation



Parse Tree

Example: Derivation

S→A1B
A→0A | ϵ
B→0B | 1B | ϵ Perform leftmost & Rightmost derivation.
(String: 00101)

Leftmost Derivation

S
A1B
0A1B
00A1B
001B
0010B
00101B
00101

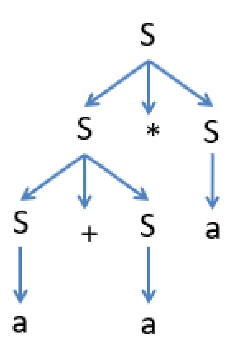
Rightmost Derivation

S
A1B
A10B
A101B
A101
0A101
00A101
00101

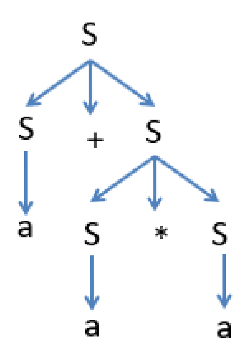
Ambiguous grammar

- Ambiguous grammar is one that produces more than one leftmost or more then one rightmost derivation for the same sentence.
- Grammar: S→S+S | S*S | (S) | a Output string: a+a*a

S
→S*S
→S+S*S
→a+S*S
→a+a*S
→a+a*a



S
→S+S
→a+S
→a+S*S
→a+a*S
→a+a*a



Here, ***Two leftmost derivation*** for string a+a*a is possible hence, above grammar is ambiguous.

Unambiguous grammar

Grammar: S→ S+S | S*S | (S) | a

Output string: a+a*a

Equivalent unambiguous grammar is

S→ S + T | T
T→ T * F | F
F→ (S) | a

Equivalent unambiguous grammar

S
→ S+T
→ T+T
→ F+T
→ a+T
→ a+T*F
→ a+F*F
→ a+a*F
→ a+a*a

Not possible????

Here, ***two left most derivation is not possible*** for string a+a*a hence, grammar is unambiguous.

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.

Parse tree follows the precedence of operators. The deepest sub-tree traversed first. So, the operator in the parent node has less precedence over the operator in the sub-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.

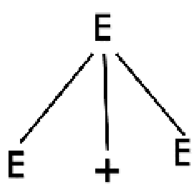
Example 1:

Production rules:

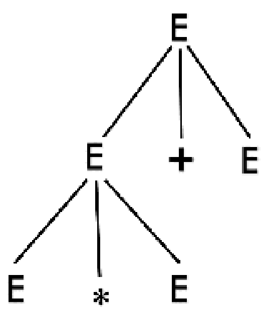
1. E = E + E
2. E = E * E
3. E = a | b | c

Input : a * b + c

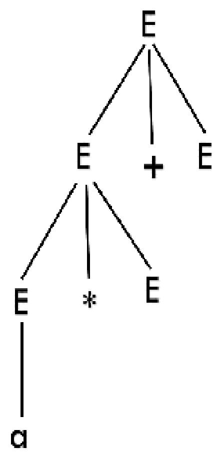
Step 1:



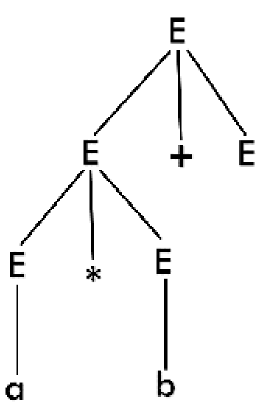
Step 2:



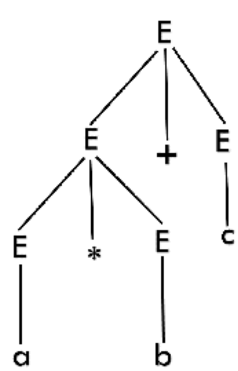
Step 2:



Step 4:



Step 5:

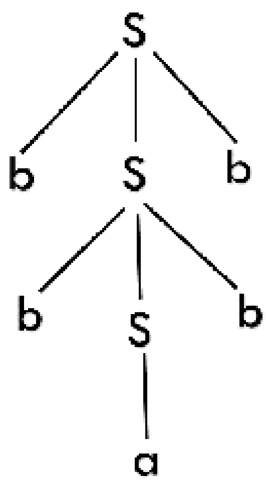


Example 2:

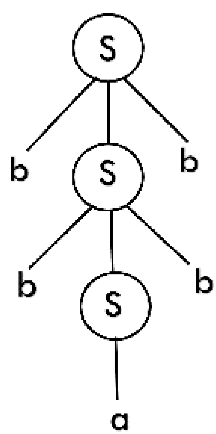
Draw a derivation tree for the string "bab" from the CFG given by

1. S → bSb | a | b

Now, the derivation tree for the string "bbabb" is as follows:



The above tree is a derivation tree drawn for deriving a string bbabb. By simply reading the leaf nodes, we can obtain the desired string. The same tree can also be denoted by,



Chomsky Normal Form (CNF)

CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

- Start symbol generating ε. For example, A → ε.
- A non-terminal generating two non-terminals. For example, S → AB.
- A non-terminal generating a terminal. For example, S → a.

Where S,A & B are non-terminal and a is terminal.

Converting CFG to CNF

- Steps to convert CFG to CNF
  1. Eliminate ^-Productions.
  2. Eliminate Unit Productions.
  3. Restricting the right side of productions to single terminal or string of two or more nonterminals.
  4. Final step of CNF. (shorten the string of NT to length 2)

Example: CFG to CNF

S→AAC

A→aAb|^

C→aC|a

Step 1: Elimination of ^ production

Eliminate A→^

S→AAC| AC | C

A→aAb| ab

C→aC|a

Step-2: Eliminate Unit Production

Unit Production is S→C

S→AAC|AC| aC|a

A→aAb|ab

C→aC|a

Step 3: Replace all mixed string with solid NT

S→AAC|AC|PC|a

A→aAb|ab

C→ aC|a

P→a

Q→b

Example: CFG to CNF

S→aAbB

A→Ab|b

B→Ba|a

Step 1 and 2 are not required as there is no ^ and unit productions

Step-3: Replace all mixed string with solid NT

S→PAQB

A→AQ|b

B→BP|a

P→a

Q→b

Step-4 : final step of CNF

S→PT1

T1→AT2

T2→QB

A→AQ|b

B→BP|a

P→a

Q→b

Example: CFG to CNF

S→AA

A→B|BB

B→abB|b|bb

Step 1 is not required as there is no ^ productions

Step-2: Eliminate Unit Production:

S→AA

A→ abB|b|bb|BB

B→abB|b|bb

Step-3:Replace all mixed string with solid NT:

S→AA

A→ PQB|b|QQ|BB

B→ PQB|b|QQ

P→a

Q→b

Step-4 : Shorten the string of NT to length 2

S→AA

A→ PT1|b|QQ|BB   T1→QB

B→ PV1|b|QQ   V1→QB

P→a

Q→b

Example: CFG to CNF

S→ASB|^

A→aAS|a

B→SbS|A|bb

Step-1: Eliminate ^-Production:

S→ASB|AB

A→aAS|a|aA

B→SbS|A|bb|bS|Sb|b

Step-2: Eliminate Unit Production:

S→ASB|AB

A→aAS|a|aA

B→SbS|aAS|a|aA|bb|bS|Sb|b

Step-3:Replace all mixed string with solid NT:

S→ASB|AB

A→PAS|a|PA

B→SQS|PAS|a|PA|QQ|QS|SQ|b

P→a

Q→b

Step-4 : Shorten the string of NT to length 2

S→AB|AT1   T1→SB

A→a|PA|PU1   U1→AS

B→ SV1|PV2|a|PA|QQ|QS|SQ|b

V1→QS   V2→AS

P→a

Q→b

Greibach Normal Form (GNF)

GNF stands for Greibach normal form. A CFG(context free grammar) is in GNF (Greibach normal form) if all the production rules satisfy one of the following conditions:

- A start symbol generating ε. For example, S → ε.
- A non-terminal generating a terminal. For example, A → a.
- A non-terminal generating a terminal which is followed by any number of non-terminals. For example, S → aASB.

Example:

1. G1 = {S → aAB | aB, A → aA| a, B → bB | b}
2. G2 = {S → aAB | aB, A → aA | ε, B → bB | ε}

The production rules of Grammar G1 satisfy the rules specified for GNF, so the grammar G1 is in GNF. However, the production rule of Grammar G2 does not satisfy the rules specified for GNF as A → ε and B → ε contains ε(only start symbol can generate ε). So the grammar G2 is not in GNF.

Steps for converting CFG into GNF

Step 1: Convert the grammar into CNF.

If the given grammar is not in CNF, convert it into CNF. You can refer the following topic to convert the CFG into CNF: Chomsky normal form

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

If the context free grammar contains left recursion, eliminate it. You can refer the following topic to eliminate left recursion: Left Recursion

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

If any production rule in the grammar is not in GNF form, convert it.

Example:

1. S → XB | AA
2. A → a | SA
3. B → b
4. X → a

As the given grammar G is already in CNF and there is no left recursion, so we can skip step 1 and step 2 and directly go to step 3.

The production rule A → SA is not in GNF, so we substitute S → XB | AA in the production rule A → SA as:

S → XB | AA
A → a | XBA | AAA
B → b
X → a

The production rule S → XB and B → XBA is not in GNF, so we substitute X → a in the production rule S → XB and B → XBA as:

S → aB | AA
A → a | aBA | AAA
B → b
X → a

Now we will remove left recursion (A → AAA), we get:

S → aB | AA
A → aC | aBAC
C → AAC | ε
B → b
X → a

Now we will remove null production C → ε, we get:

S → aB | AA
A → aC | aBAC | a | aBA
C → AAC | AA
B → b
X → a

The production rule S → AA is not in GNF, so we substitute A → aC | aBAC | a | aBA in production rule S → AA as:

S → aB | aCA | aBACA | aA | aBA
A → aC | aBAC | a | aBA
C → AAC
C → aCA | aBACA | aA | aBAA
B → b
X → a

The production rule C → AAC is not in GNF, so we substitute A → aC | aBAC | a | aBA in production rule C → AAC as:

1. S → aB | aCA | aBACA | aA | aBAA
2. A → aC | aBAC | a | aBA
3. C → aCAC | aBACAC | aAC | aBAAC
4. C → aCA | aBACA | aA | aBAA
5. B → b
6. X → a

Hence, this is the GNF form for the grammar G.