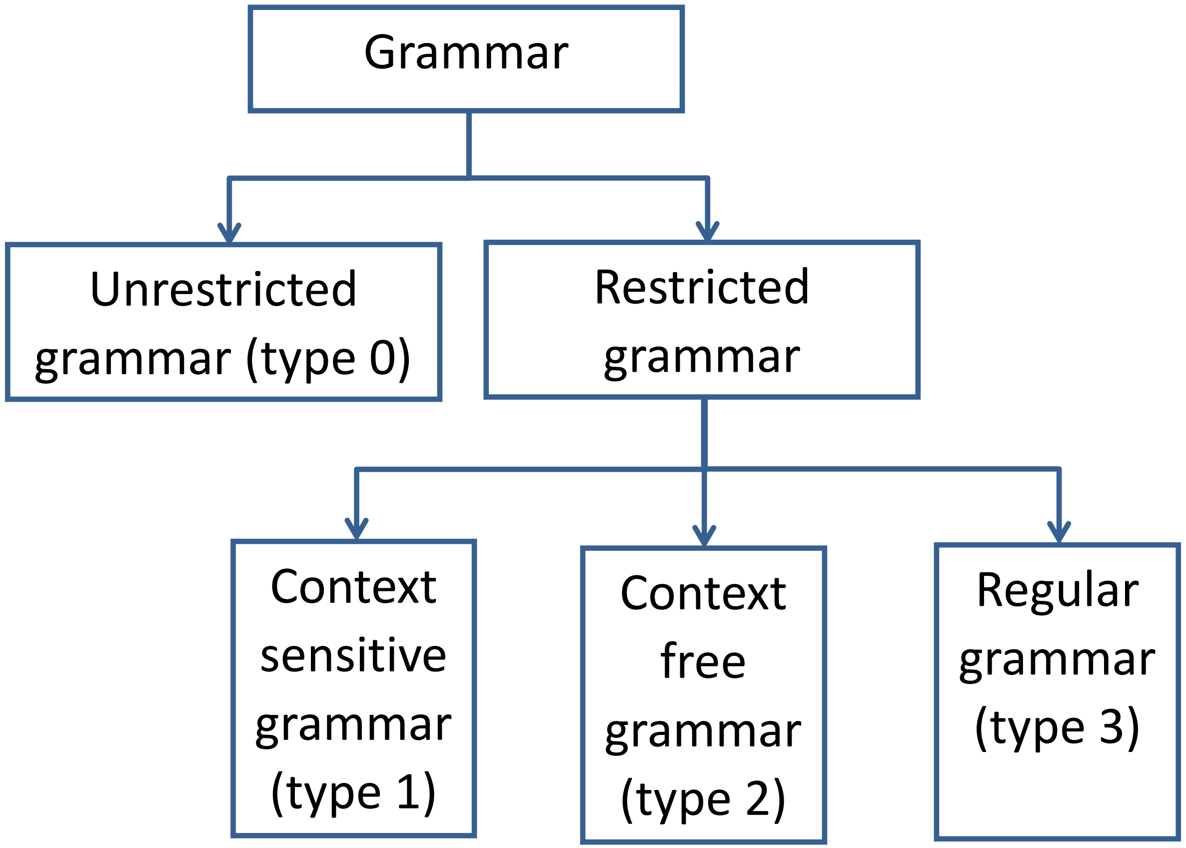
Context Free Grammar

Chomsky Hierarchy

Chomsky hierarchy (Classification of grammar)



Type 0 grammar (Phrase Structure Grammar)

- Their productions are of the form:

$$\alpha \rightarrow \beta$$

- α is (V+T)* V (V+T)* β is (V+T)*

  V=Variable / Non-Terminal

  T=Terminal

- It is also known as **unrestricted grammar.**
- Example: S → ACaB

  Bc → acB

  CB → DB

  aD → Db

  S → ε

Type 1 grammar (Context Sensitive Grammar)

- Their productions are of the form:

$$\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}$$

- Where $|\boldsymbol{\alpha}| \leq |\boldsymbol{\beta}|$
- The count of symbol in $\boldsymbol{\alpha}$ is less than or equal to $\boldsymbol{\beta}$.
- Example: AB → AbBc

  A → bcA

  B → b

Type 2 grammar (Context Free Grammar)

- Their productions are of the form:

$$\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}$$

- Where $|\alpha| = 1$ *and there is no restriction on* β .
- Example: S → Xa

  X → a

  X → aX

  X → abc

Type 3 grammar (Regular grammar)

- Their productions are of the form:

$$\mathbf{V} \rightarrow \boldsymbol{VT}^{*} \mid \boldsymbol{T}^{*} \text{ or } \quad \boldsymbol{V} \rightarrow \boldsymbol{T}^{*}\boldsymbol{V} \mid \boldsymbol{T}^{*}$$

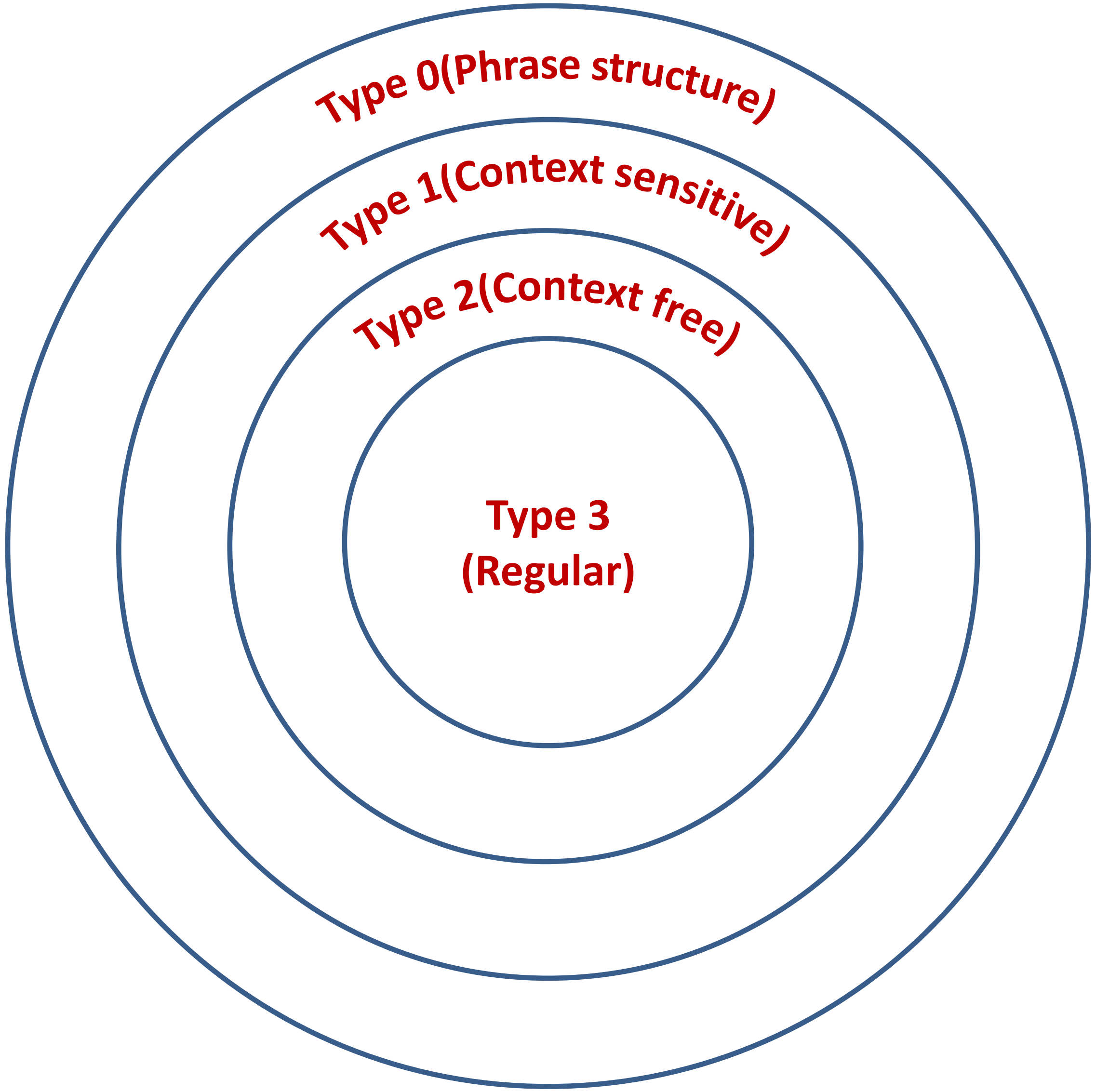
- Example: X → a | aY

  Y → b

Summary

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

Hierarchy of grammar



Grammar

Grammar

- A grammar is a 4-tuple $G = (N, T, S, P)$ where,

  N is finite set of non terminals,

  T is finite set of terminals,

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

  P is a finite set of productions

- Grammar G1 – ({S, C, D}, {c, d}, S, {S → CD, C → c, D → d})
  - N={S,C,D}
  - T={c,d}
  - S=S
  - P= S → CD, C → c, D → d

Derive String from Grammar

- Grammar $G = (\{S, C\}, \{c, d\}, S, \{S \rightarrow cCd, cC \rightarrow ccCd, C \rightarrow \varepsilon\})$

S ⇒ cCd using production **S →cCd**

⇒ccCdd using production **cC →cCd**

⇒cccCddd using production **cC→ cCd**

⇒cccddd using production **C→ ε**

Context Free Grammar

Context Free Grammar

- A context free grammar (CFG) is a 4-tuple $G = (N, T, S, P)$ where,

  N is finite set of non terminals,

  T is finite set of terminals,

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

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

Context Free Grammar

- Application of CFG:
    1. CFG are extensively used to specify the syntax of programming language.
    2. CFG is used to develop a parser.

Generating Strings with CFGs

- Start with the initial symbol
- Repeat:
  - Pick any non-terminal in the string
  - Replace that non-terminal with the right-hand side of some rule that has that non-terminal as a left-hand side
- Until all elements in the string are terminals

Generating Strings with CFGs

- S → aS
- S → Bb
- B → cB
- B → Ɛ

Generating a string:

S	replace S with aS
aS	replace S with Bb
aBb	replace B with cB
acBb	replace B with Ɛ
acb	Final String

Generating Strings with CFGs

- S → aS
- S → Bb
- B → cB
- B → Ɛ

Generating a string:

S	replace S with aS
aS	replace S with aS
aaS	replace S with Bb
aaBb	replace B with cB
aacBb	replace B with cB
aaccBb	replace B with Ɛ
aaccb	Final String

Generating Strings with CFGs

- S → aS | Ɛ

  Possible strings={Ɛ, a, aa, aaaa, aaaa, ….}

- S → bS | Ɛ

  Possible strings={Ɛ, b, bb, bbb, bbbb, ……}

- S → aS | bS | Ɛ

  Possible strings={Ɛ, a, b, aa, bb, ab, aba, bbab, …..}

CFG Examples

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

  $S \rightarrow a \mid b$

- **Write CFG for a^+**

  $S \rightarrow aS \mid a$

- **Write CFG for a***

  $S \rightarrow aS \mid \wedge$

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

  $S \rightarrow abS \mid \wedge$

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

  $S \rightarrow aS \mid bS \mid a \mid b$

Generating Strings with CFGs

- $E \to E + E \mid E - E \mid a \mid b$

Derive string “a-b+a”

Generating a string:

E	E
E-E	E+E
a-E	E-E+E
a-E+E	a-E+E
a-b+E	a-b+E
a-b+a	a-b+a

Generating Strings with CFGs

- S→0S1S | 1S0S | Ɛ

Derive string “011100”

Generating a string:

S

0S1S

01S

011S0S

0111S0S0S

01110S0S

011100S

011100

Derivation

Derivation

- The process of deriving a string is called as **derivation**.
- 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

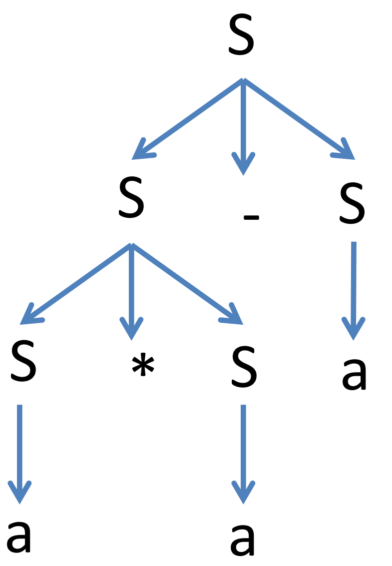
→**a***S-S

→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**

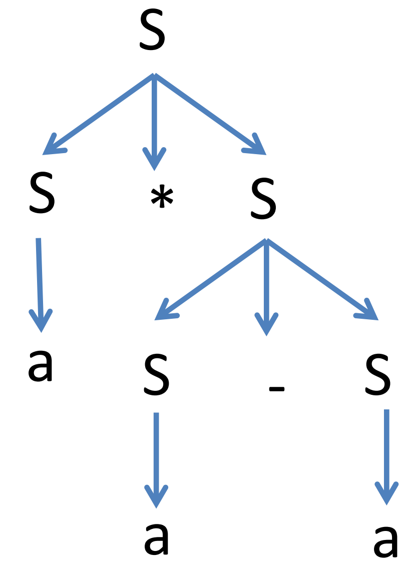
→S***S-S**

→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

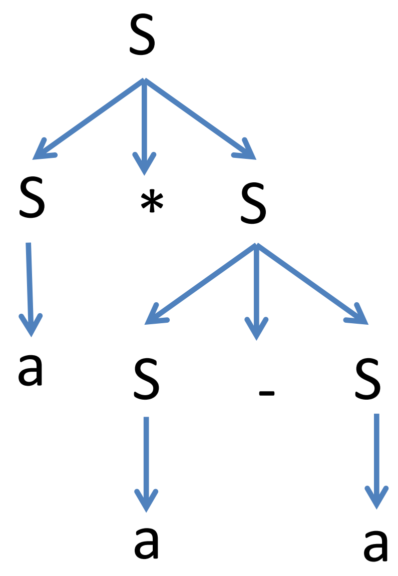
0A101

00A101

00101

Parse Tree

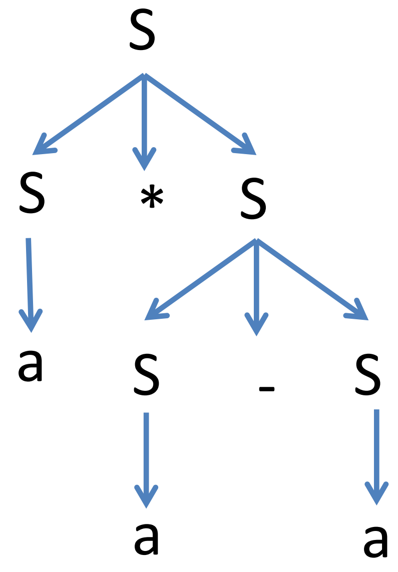
- The graphical representation of a derivation is called as a **parse tree** or **derivation tree**.
- Grammar: S→S+S | S-S | S*S | S/S | a   Output string: a*a-a



Parse Tree

Parse Tree

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



Parse Tree

Exercise: Derivation

Perform

1) Leftmost derivation and rightmost derivation.
2) Draw parse tree.

$S \rightarrow A1B$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow 0B \mid 1B \mid \epsilon$

Output string: 1001.

Ambiguous grammar

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

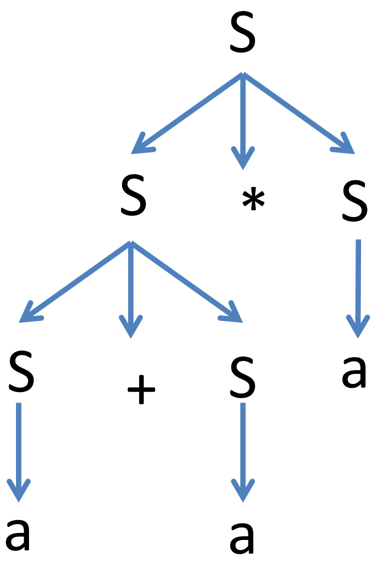
→**S*S**

→**S+S***S

→**a**+S*S

→a+**a***S

→a+a***a**



S

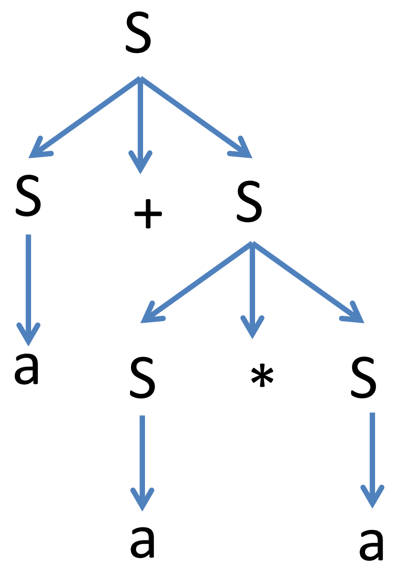
→**S+S**

→**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.

Exercise: Ambiguous grammar

Check whether following grammar is ambiguous or not:

1. E→ I
2. E→ E +E
3. E → E * E
4. E → (E)
5. I → 0 | 1 | 2 | … |9 | ϵ

String: $3*2+5$

Unambiguous grammar

Grammar: S→ S+S | S*S | (S) | a

Equivalent unambiguous grammar is

S→ S + T | T
T→ T * F | F
F→ (S) | a

Equivalent unambiguous grammar

Not possible????

Output string: a+a*a

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

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

Inherently Ambiguous

- If every grammar that generates Language L is ambiguous then Language L is called as Inherently Ambiguous Language.
- If a grammar is ambiguous, it does not imply that its language will be ambiguous too.
- If a grammar is ambiguous, its language may be unambiguous.
- If a grammar is ambiguous, its language will be unambiguous when there exists at least one unambiguous grammar which generates that language.

Simplified forms & Normal forms

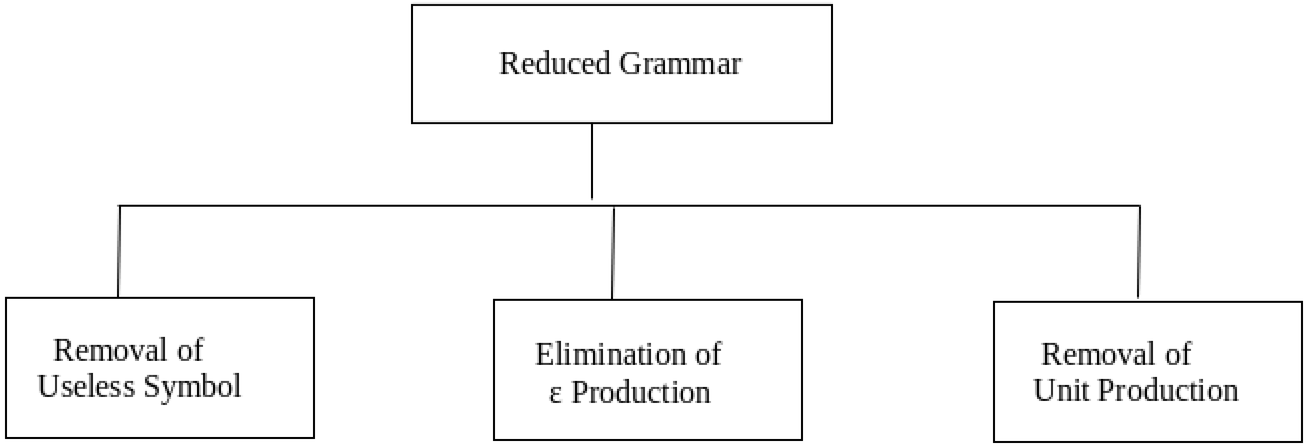
Simplification of CFG

- The definition of context free grammars (CFGs) allows us to develop a wide variety of grammars.
- Most of the time, some of the productions of CFGs are not useful and are redundant. This happens because the definition of CFGs does not restrict us from making these redundant productions.
- By simplifying CFGs we remove all these redundant productions from a grammar , while keeping the transformed grammar equivalent to the original grammar.
- All the grammar are not always optimized that means the grammar may consist of some extra symbols(non-terminal). Having extra symbols, unnecessary increase the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols.

Simplification of CFG

The properties of reduced grammar are given below:

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



Nullable Variable

- A Nullable variable in a CFG, $G = (N, T, S, P)$ is defined as follows:
  1. Any variable A for which P contains $A \rightarrow$ ^ is nullable.
  2. If P contains the production $A \rightarrow B_1\ B_2 \ldots . Bn$ are nullable variable, then A is nullable.
  3. No other variables in V are nullable.

Eliminate ^ production

S→a X |Yb
X→ ^ | S
Y→bY|b

Nullable variable={X}

S→aX | Yb | a^
X→^ | S
Y→bY|b

Replacing X by ^ in all productions containing X on RHS and rewriting the production again

S→aX|Yb|a
X→S
Y→bY|b

Removing ^ productions

Exercise: Eliminate ^ production

S→AC

A→aAb|^

C→aC|a

After elimination of ^ production:

S→AC | C

A→aAb| ab

C→aC|a

S→XaX|bX|Y

X→XaX|XbX|^

Y→ab

After elimination of ^ production:

S→ XaX | bX | Y | aX | Xa | a | b

X→ XaX |XbX | aX | Xa | a | Xb | bX | b

Y→ab

Unit Production

- A production of the form $A \rightarrow B$ is termed as unit production. Where A & B are nonterminals.

Elimination of unit production

S→ABA|BA|AA|AB|A|B
A→ aA|a
B→ bB|b

Unit Productions are S→A and S→B

S→ABA|BA|AA|AB |aA|a |bB|b
A→aA|a
B→bB|b

Removing unit productions

Useless Symbols

- A symbol can be useless if it does not appear on the right-hand side of the production rule and does not take part in the derivation of any string. That symbol is known as a useless symbol.

Elimination of Useless Symbols

S→abS | abA | abB
A→ cd
B→aB
C→dc

S→abS | abA
A→ cd
C→dc

S→abS | abA
A→ cd

Production 'B->aB' is useless because there is no way it will ever terminate .

production 'C->dc' is useless because the variable 'C' will never occur in derivation of any string.

Elimination of Useless Symbols

T→aaB | abA | aaT
A→ aA
B→ab|b
C→ad

T→aaB | aaT
B→ ab|b
C→ad

T→aaB | aaT
B→ ab|b

Production ‘A->aA’ is useless because there is no way it will ever terminate .

production ‘C->ad’ is useless because the variable ‘C’ will never occur in derivation of any string.

Exercise

- S→AB

A→ aAb

A→ab

B→bB

B→b

C→cCd

C→cd

CFG to CNF

Chomsky Normal Form (CNF)

- A context free grammar is in Chomsky normal form (CNF) if every production is one of these two forms:

$$A \rightarrow BC$$

$$A \rightarrow a$$

Where $A, B,$ and C are nonterminal 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

$S \rightarrow AAC$

$A \rightarrow aAb \mid \wedge$

$C \rightarrow aC \mid a$

Step 1: Elimination of ^ production

Eliminate $A \rightarrow \wedge$

$S \rightarrow AAC \mid AC \mid C$

$A \rightarrow aAb \mid ab$

$C \rightarrow aC \mid a$

Step-2: Eliminate Unit Production

Unit Production is $S \rightarrow C$

$S \rightarrow AAC \mid AC \mid aC \mid a$

$A \rightarrow aAb \mid ab$

$C \rightarrow aC \mid a$

Step 3: Replace all mixed string with solid NT

$S \rightarrow AAC \mid AC \mid PC \mid a$

$A \rightarrow PAQ \mid PQ$

$C \rightarrow PC \mid a$

$P \rightarrow a$

$Q \rightarrow b$

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

$S \rightarrow AX_1$ $\quad X_1 \rightarrow AC$

$S \rightarrow AC \mid PC \mid a$

$A \rightarrow PY_1$ $\quad Y_1 \rightarrow AQ$

$A \rightarrow PQ$

$C \rightarrow PC \mid a$

$P \rightarrow a$

$Q \rightarrow b$

Chomsky Normal Form

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→ PT1|b|QQ

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)

A context free grammar is in Greibach normal form (GNF) if every production is one of these two forms:

- **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.
- Greibach Normal Form is useful for **proving the equivalence of cfgs and npdas.** When we discuss converting a cfg to an npda, or vice versa, we will use Greibach Normal Form.

Greibach Normal Form (GNF)

G1 = {S → aAB | aB, A → aA| a, B → bB | b}

- Is it in GNF?
- Yes

G2 = {S → aAB | aB, A → aA | ε, B → bB | ε}

- Is it in GNF?
- No

Left recursion

A grammar is said to be left recursive if it has a non terminal A such that there is a derivation $A \rightarrow A\alpha$ for some string α.

Algorithm to eliminate left recursion

1. Arrange the non terminals in some order $A_1, \dots, An$
2. for $i := 1\ to\ n$ **do begin**

   for $j := 1\ to\ i - 1$ **do begin**

   replace each production of the form $A_i \rightarrow Ai\gamma$

   by the productions $A_i \rightarrow \delta_1\gamma |\ \delta_2\gamma | \dots\dots |\ \delta_k\gamma$,

   where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots . \mid \delta_k$ are all the current A_j productions;

   end

   eliminate the immediate left recursion among the A_i - productions

end

Left recursion elimination

$A \rightarrow A\alpha \mid \beta$ ⟶ $A \rightarrow \quad A'$

$A' \rightarrow \quad A' \mid \epsilon$

Examples: Left recursion elimination

$E \rightarrow E+T \mid T$

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow T*F \mid F$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$X \rightarrow X\%Y \mid Z$

$X \rightarrow ZX'$

$X' \rightarrow \%YX' \mid \varepsilon$

Examples: Left recursion elimination

S→ Aa | b

A→ Ac | Sd | ε

Here, Non terminal S is left recursive because:

S→ Aa → Sda

Aad | bd

S→ Aa | b

A→ Ac

A→ Sd

A→ ε

A→Ac | Aad | bd | ε

To remove indirect left recursion replace S with productions of S

Now, remove left recursion

S→ Aa | b
A→Ac | Aad | bd | ε

S→ Aa | b

A→ bdA' | A'

A'→ cA' | adA' | ε

Exercise

1. A→Abd | Aa | a

   B→Be | b

2. A→AB | AC | a | b

Converting CFG to GNF

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

If the given grammar is not in CNF, convert it into CNF.

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

If the context free grammar contains left recursion, eliminate it.

- **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: CFG to GNF

$S \rightarrow aBc$

$B \rightarrow b$

Is it in GNF?

NO

Convert it in GNF

$S \rightarrow aBC$

$B \rightarrow b$

$C \rightarrow c$

Grammar in GNF Format

Example: CFG to GNF

S → XB | AA

A → a | SA

B → b

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

Example: CFG to GNF

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

Remove left recursion (A → AAA)

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

Remove null production C → ε

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

Example: CFG to GNF

S → aB | aCA | aBACA | aA | aBAA
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:

S → aB | aCA | aBACA | aA | aBAA
A → aC | aBAC | a | aBA
C → aCAC | aBACAC | aAC | aBAAC
C → aCA | aBACA | aA | aBAA
B → b
X → a

Example: CFG to GNF

$S \rightarrow Ab$
$A \rightarrow aS$
$A \rightarrow a$

$S \rightarrow aSb \mid ab$
$A \rightarrow aS$
$A \rightarrow a$

$S \rightarrow aSB \mid aB$
$A \rightarrow aS$
$A \rightarrow a$
$B \rightarrow b$

Is it in GNF?

No

$S \rightarrow aSB \mid aB$
$\mathbf{A \rightarrow aS}$
$\mathbf{A \rightarrow a}$
$B \rightarrow b$

Useless Symbols

Why?

$S \rightarrow aSB \mid aB$
$B \rightarrow b$

GNF

Decision Properties of CFG

- Is a given string in a CFL?
- Is a CFL empty?

Decision Properties of CFG

- **Is a given string in a CFL?**

1) If we are given the CFL as a PDA, we can answer this simply by executing the PDA.

2) If we are given the language as a grammar, we can either

- Convert the grammar to a PDA and execute the PDA, or
- Convert the grammar to Chomsky Normal Form and parse the string to find a derivation for it.

Decision Properties of CFG

- **Is a CFL empty?**
  - Detect whether a variable is nullable.
  - Determine if the grammar's start symbol is nullable.

Decision Properties of CFG

No algorithm exists to determine if

- Two CFLs are the same.
  - Note that we *were* able to determine this for regular languages.
- Two CFLs are disjoint (have no strings in common).

Union, Concatenation & Kleene's of CFG

Union, Concatenation & Kleene's of CFG

Theorem:- If L_1 and L_2 are context - free languages, then the languages L_1 U L_2, L_1L_2 , and L_1^* are also CFLs.

Union

- If L1 and If L2 are two context free languages, their union L1 ∪ L2 will also be context free.
- L1 = { $a^n b^n c^m$ | m >= 0 and n >= 0 } and L2 = { $a^n b^m c^m$ | n >= 0 and m >= 0 }
  L3 = L1 ∪ L2 = { $a^n b^n c^m \cup a^n b^m c^m$ | n >= 0, m >= 0 }
- L3 is also Context Free Language.

Concatenation

- If L1 and If L2 are two context free languages, their union L1 . L2 will also be context free.
- L1 = { $a^n b^n$ | n >= 0 } and L2 = { $c^m d^m$ | m >= 0 }
  L3 = L1.L2 = { $a^n b^n c^m d^m$ | m >= 0 and n >= 0}
- L3 is also Context Free Language.

Intersection and complementation

- If L1 and If L2 are two context free languages, their intersection L1 ∩ L2 need not be context free.

Closure Property Summary

Context-free languages are **closed** under −

- Union
- Concatenation
- Kleene Star operation

Context-free languages are **not closed** under −

- Intersection
- Complement

End of Unit - 3