FORMAL LANGUAGES AND AUTOMATA THEORY

Module 3

Rijin IK

Assistant Professor
Department of Computer Science and Engineering
Vimal Jyothi Engineering College
Chemperi

October 9, 2023

Outline

Course Outcomes

After the completion of the course the student will be able to

1. Classify a given formal language into Regular, Context-Free, Context Sensitive, Recursive or Recursively Enumerable. [Cognitive knowledge level: Understand]
2. Explain a formal representation of a given regular language as a finite state automaton, regular grammar, regular expression and Myhill-Nerode relation. [Cognitive knowledge level: Understand]
3. Design a Pushdown Automaton and a Context-Free Grammar for a given context-free language. [Cognitive knowledge level : Apply]
4. Design Turing machines as language acceptors or transducers. [Cognitive knowledge level: Apply]
5. Explain the notion of decidability. [Cognitive knowledge level: Understand]

Myhill-Nerode Relations (MNR)

Equivalence relation

- Let $R \subseteq \Sigma^*$ be a regular set, and let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for R with no inaccessible states. The automaton M induces an Equivalence relation $\equiv_M$ on Σ^* defined by

$$x \equiv_M y \overset{def}{\iff} \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y).$$

[Equivalence relation$\rightarrow$ it is reflexive, symmetric, and transitive]

- **Reflexive**
  - $\forall_x \in \Sigma^*, \hat{\delta}(q_0, x) = \hat{\delta}(q_0, x)$
- **Symmetric**
  - $x \equiv_M y \implies \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) \implies \hat{\delta}(q_0, y) = \hat{\delta}(q_0, x) \implies y \equiv_M x$
- **Transitive**
  - suppose $x \equiv_M y \; and \; y \equiv_M z \implies x \equiv_M z$

Myhill-Nerode Relations (MNR)

- The Myhill-Nerode relation for a regular set $R \subseteq \Sigma^*$ is an equivalence relation $\equiv$ on Σ^* that satisfies three key properties:
  1. It is a right congruence: for any $x, y \in \Sigma^*$ and $a \in \Sigma$,
     - $x \equiv_m y \implies xa \equiv_m ya$;
  2. It refines $L(M)$: for any $x, y \in \Sigma*$
     - $x \equiv_m y \implies (x \in L(M) \Leftrightarrow y \in L(M))$;
     - i.e whenever $x \& y$ are related either $x \& y$ are in the language or both $x \& y$ are not in the language.
  3. It is of finite index; that is, it has only finitely many equivalence classes.
     - This is beause there is exactly one equivalence class $x \in \Sigma^* \middle| \hat{\hat{\delta}}(q_0, x) = q$ corresponding to eaeh state q of M.

Myhill-Nerode Relations (MNR)

Formal definition of Myhill-Nerod Relation (MN relation)

Let L be a language over an alphabet set Σ.Then, an MN relation $\equiv$ is an equivalence relation on Σ^*, which is a right congruence of finite index refining L.

Example:

- $L = x \in \{a^* \big| 3 \ divides \ |x|\}$
- $x \equiv y \Leftrightarrow |x| mod3 = |y| mod \ 3$. Equivalence classes are:
  - $[\epsilon] = \{a^i \big| i \mod 3 = 0\}$
  - $[a] = \{a^i \big| i \mod 3 = 1\}$
  - $[a^2] = \{a^i \big| i \mod 3 = 2\}$

Conversion of DFA to MNR

Lemma

If a language L is regular, then there is a Myhill-Nerode relation on Σ^* with respect to L

Proof

L is Regular $\implies$ $\exists$ a DFA M such that L=L(M)

Consider a binary relation $\equiv_M$ on Σ^* induced by$M = (Q, \Sigma, \delta, q_0, F)$, defined as

$$\forall_{x,y} \in \Sigma^*, x \equiv_M y \Leftrightarrow \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$$

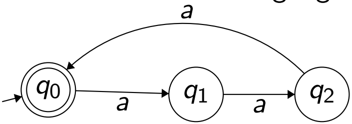
we proved that $\equiv_M$ is an MN relation

Conversion of DFA to MNR

Q:Show the equivalence classes of Myhill Nerod relation for the language $L = \{x \in \{a\}^* \middle| 3 \ divides \ |x|\}$.

Soln:

DFA for the above language



The DFA has 3 states. so MNR partions the Σ^* into 3 classes:

- $C_1 = \{x \in \Sigma^* \middle| \hat{\delta}(q_0, x) = q_0\}$ $[q_0] = \{a^i \middle| i \mod 3 = 0\}$
  - $c_1 = \{\epsilon, a^3, a^6 ....\}$
- $C_2 = \{x \in \Sigma^* \middle| \hat{\delta}(q_0, x) = q_1\}$ $[q_1] = \{a^i \middle| i \mod 3 = 1\}$
  - $c_2 = \{a^1, a^4 ....\}$
- $C_3 = \{x \in \Sigma^* \middle| \hat{\delta}(q_0, x) = q_2\}$ $[q_2] = \{a^i \middle| i \mod 3 = 2\}$
  - $c_3 = \{a^2, a^5 ....\}$

Conversion of MNR to DFA

- Given an MN relation $\equiv$ for a Language L over an alphabet set Σ, one can automatically construct a DFA $M_{\equiv} = (Q, \Sigma, \delta, q_0, F)$, such that $L(M_{\equiv}) = L$.

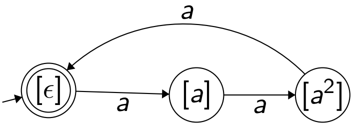
$$\begin{aligned}
Q &= \{[x] \big| x \in \Sigma^*\} \;\; \textit{set of all equivalent class} \\
q_0 &= [\epsilon] \\
\delta([x], a) &= [xa] \\
F &= \{[x] \big| x \in L\}
\end{aligned}$$

Conversion of MNR to DFA

Example: Consider the MN relation represented by the following equivalence classes for the language : $L = \{x \in \{a\}^* \big| 3 \textit{ divides } |x|\}$.
Equivalence classes are:

- $[\epsilon] = \{a^i \big| i \mod 3 = 0\}$
- $[a] = \{a^i \big| i \mod 3 = 1\}$
- $[a^2] = \{a^i \big| i \mod 3 = 2\}$

$$\begin{aligned} Q &= \{[x] \big| x \in \Sigma^*\} \\ q_0 &= [\epsilon] \\ \delta([x], a) &= [xa] \\ F &= \{[x] \big| x \in L\} \end{aligned}$$



The Myhill–Nerode Theorem

- Let $R \subseteq \Sigma^*$. The following statements are equivalent:
  1. R is regular;
  2. There exists a Myhill-Nerode relation for R:
  3. The relation $\equiv_R$ is of finite index.

Applications of MNT

- Proving whether a language is regular or not.
- It can be also used to find the minimal number of states in a Deterministic Finite Automata (DFA).

Example: Is the following language regular

$$L = \{x \in \{a,b\}^* \,\big|\, \text{number of a in x is odd}\}$$

Sln:

- It is enough to prove that $\equiv_L$ is of finite index
- Recall that: $\forall_{x,y} \in \Sigma^*, x \equiv_L y \ iff \ \forall_z \in \Sigma^*, (xz \ \in L \ \Leftrightarrow yz \in L)$
- Equivalence classes of $\equiv_L$:
  - Suppose $x, y, z \in \{a,b\}^*$ such that number of a in x is even and number of a in y is even:
  - **Case 1:** number of a in z is odd

$$\Longrightarrow \quad \text{number of a in (xz) is odd and number of a in (yz) is odd}$$
$$\Longrightarrow \quad xz \in L \ and \ yz \in L$$

  - **Case 2:** number of a in z is even

$$\Longrightarrow \quad \text{number of a in (xz) is even and number of a in (yz) is even}$$
$$\Longrightarrow \quad xz \notin L \ and \ yz \notin L$$

so the equivalent class is $[\epsilon] = \{x \in \{a,b\}^* \,\big|\, \text{number of a in z is even}\}$

Sln cont..:

- Suppose $x, y, z \in \{a, b\}^*$ such that number of a in x is odd and number of a in y is odd:
- **Case 1:** number of a in z is odd

$$\implies \quad \text{number of a in (xz) is even and number of a in (yz) is even}$$
$$\implies \quad xz \notin L \; and \; yz \notin L$$

- **Case 2:** number of a in z is even

$$\implies \quad \text{number of a in (xz) is odd and number of a in (yz) is odd}$$
$$\implies \quad xz \in L \; and \; yz \in L$$

so the equivalent clas is $[a] = \{x \in \{a, b\}^* \big| \text{number of a in z is odd}\}$

Equivalence classes of $\equiv_L$:

1. $[\epsilon] = \{x \in \{a, b\}^* \big| \text{number of a in z is even}\}$
2. $[a] = \{x \in \{a, b\}^* \big| \text{number of a in z is odd}\}$

Thus $\equiv_L$ is of finite index and hence the language is regular

Context Free Grammars

- Context-free grammars (CFGs) are used to describe context-free languages.
- A context-free grammar is a set of recursive rules used to generate patterns of strings.
- A context-free grammar can describe all regular languages and more, but they cannot describe all possible languages.
  - i.e It is **more powerfull than Regular grammar**
- Widely used in Programming Languages-syntax,parsing.

Applications of Context Free Grammar (CFG)

- For defining programming languages
- For parsing the program by constructing syntax tree
- For translation of programming languages
- For describing arithmetic expressions
- For construction of compilers
- Document Type Definition in XML is a Context Free Grammars which describes the HTML tags and the rules to use the tags in a nested fashion.

Formal definition of Context Free Grammars

- A context-free grammar (CFG) is denoted $G = (V, T, P, S)$, where
  - V and T are finite sets of variables and terminals, respectively.
  - P is a finite set of productions; each production is of the form $A \rightarrow \alpha$,
    - where A is a variable and α is a string of symbols from $(V \cup T)^*$.
  - S is a special variable called the start symbol.

Language generated by CFG

- The notation $S \overset{*}{\underset{G}{\Longrightarrow}} w$ represents the fact that we can derive the string w from the start symbol of the grammar G in zero or more steps
- The language generated by G [denoted $L(G)$] is $\{w|\ w\ is\ inT^*\ and\ S \overset{*}{\underset{G}{\Longrightarrow}} w\}$. That is, a string is in L(G) if:
  - The string consists solely of terminals.
  - The string can be derived from S.

*A language A is said to be a context free language if there exsits a context free grammar G such that $A = L(G)$
*The production rules are context-independent, meaning the replacement of non-terminal symbols can occur without regard to the surrounding context.

C language is an example for Context Free Language.

Example:

- Write CFG for the language $L = \{a^n b^n \middle| n \geq 1\}$

$$\begin{aligned} L &= \{ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, \ldots\ldots\ldots\ldots\} \\ G &= (\{S\}, \{a, b\}, P, S) \end{aligned}$$

P:

$$\begin{aligned} S &\rightarrow aSb \middle| ab \\ &(Or) \\ S &\rightarrow aSB \\ S &\rightarrow aB \\ B &\rightarrow b \end{aligned}$$

Proving correctness of CFGs

- $L \subseteq L(G)$: Every string in L can be generated by G.
- $L(G) \subseteq L$: G only generate strings of L.

Example:

$$\begin{array}{rcl} G & = & (V, \Sigma, S, P) \\ V & = & \{S\} \\ \Sigma & = & \{0, 1\} \\ S & = & S \\ P & = & \{S \rightarrow 0S1|S1|\epsilon\} \\ & or & \\ G & : & S \rightarrow 0S1|S1|\epsilon \end{array}$$

$L(G) = L$ where $L = \{0^i 1^j || i \leq j\}$ prove $L(G) = L$?

Proving correctness of CFGs

Proof: Need to show that $L \subseteq L(G)$ and $L(G) \subseteq L$.

$\Rightarrow$ Suppose $x \in L$. Must show x can be generated by G.

Proof by induction on the length of x, $\left|x\right|$.

Basis:

- $|x| = 0$. Then $x = \epsilon$ and $S \rightarrow \epsilon$ is a rule in G.

Induction Hypothesis:

- Suppose all words in L shorter than x can be generated in the grammar and that $x \notin \epsilon$.
- Need to show x can also be generated.

Induction Step: since $x \in L, x = 0^i 1^j$

- **Case 1:** $i = 0$ *so* $x = 1^j = 1^{j-1}1$
  - Since $1^{j-1} \in L$ and $|1^{j-1}| < |x|, S \overset{*}{\Rightarrow} 1^{j-1}$ by the I.H.
  - Then use rule $S \rightarrow S1$ followed by the derivation $S \overset{x}{\Rightarrow} 1^{j-1}$ to construct x.
  - Therefore, $x \in L(G)$

Proof cont..

- **Case 2:** $i \neq 0$ *so* $x = 0^i 1^j$
  - Then $x' = 0^{i-1}1^{j-1} \in L$ since $i - 1 \leq j - 1$.
  - $|x'| < |x|$ so by the I.H., $x' \in L(G)$ and $S \stackrel{*}{\Longrightarrow} 0^{i-1}1^{j-1}$
  - Then use rule $S \rightarrow 0S1$ followed by the derivation $S \stackrel{*}{\Longrightarrow} 0^{i-1}1^{j-1}$ to construct x.
  - Thus $S \stackrel{*}{\Longrightarrow} x$ *so* $x \in L(G)$.

$\Leftarrow$Must show if $x \in L(G)$ *the* $x \ \in L$.
Proof by induction on k, the number of steps in x's derivation.
Basis:

- 1 step $(S \Rightarrow \epsilon)$: Then $x = \epsilon$ and $x \in L$.

Induction Hypothesis:

- Assume every word in L(G) with a shorter derivation than x is in L.
- Either $x = 0w1$ *or* $x = w1$ where $w \in L(G)$ and w can be derived from S in less than k steps;
  - i.e. $S \stackrel{<k}{\Longrightarrow} w$

Proving correctness of CFGs

Proof cont..

- By I.H. we know $w \in L$ so it has at least as many 1's as 0's
- Then by applying either $S \to 0S1$ *or* $S \to S1$ followed by the derivation $S \stackrel{*}{\Rightarrow}$ w we obtain x in the form 0^i1^j where we are only increasing the difference of 1's to 0's so $i \leq j$ and $x \in L$.

Therefore, $L(G) \subseteq L$ and $L \subseteq L(G)$ so $L = L(G)$.

Derivation Trees and ambiguity

ϵ –Productions

- A production of the form $A \to \epsilon$, where A is a variable, is called a null production or ϵ –Productions

Unit Productions

- A production of the form $A \to B$ whose right-hand side consists of a single variable is called a unit production.
- All other productions, including those of the form $A \to a$ and ϵ -productions, are nonunit productions.

Recursive Productions

- If a production's left side occurs in right side
- $S \to aS$ (Directly recursive)
- $S \to aA,\ A \to b|bS$ (indirectly recursive)

Derivation Trees and ambiguity

Derivation

- Derivation is the process of applying productions repeatedly to expand non-terminals in terms of terminals or non-terminals, until there are no more non-terminals.
- A derivation can be either Leftmost derivation or Right most derivation.

Leftmost derivation

- If at each step in a derivation a production is applied to the leftmost variable, then the derivation is said to be leftmost.
- **Example:**
  - Consider the grammar $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of

$$\begin{aligned} S &\rightarrow aAS|a \\ A &\rightarrow SbA|SS|ba \end{aligned}$$

  - The corresponding rightmost derivation is

$$S \implies aAS \implies aSbAS \implies aabAS \implies aabbaS \implies aabbaa.$$

Rightmost derivation:

- A derivation in which the rightmost variable is replaced at each step is said to be rightmost.
- **Example:**
  - Consider the grammar $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of

$$\begin{array}{rcl} S & \rightarrow & aAS|a \\ A & \rightarrow & SbA|SS|ba \end{array}$$

  - The corresponding rightmost derivation is

$$S \implies aAS \implies aAa \implies aSbAa \implies aSbbaa \implies aabbaa.$$

*****Note:** "If w is in L(G) for CFG G, then w has at least one parse tree, and corresponding to a particular parse tree, w has a unique leftmost and a unique rightmost derivation."

Sentential Form:

- A sentential form is any string consisting of non-terminals and/or terminals that is derived from a start symbol.
- A string of terminals and variables α is called a sentential form if $S \stackrel{*}{\Rightarrow} \alpha$

Example:

- Consider the grammar $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of

$$\begin{aligned} S &\rightarrow aAS|a \\ A &\rightarrow SbA|SS|ba \end{aligned}$$

- The corresponding rightmost derivation is
  $S \implies aAS \implies aAa \implies aSbAa \implies aSbbaa \implies aabbaa.$

$S \stackrel{*}{\Rightarrow} aAa$, $S \stackrel{*}{\Rightarrow} aSbAa$, $S \stackrel{*}{\Rightarrow} aSbbaa$,$S \stackrel{*}{\Rightarrow} aabbaa$. are Sentential Form

Derivation Trees (or) Parse tree:

- The derivations in a CFG can be represented using trees. Such trees representing derivations are called derivation trees.
- Let G = (V, T, P, S) be a CFG. A tree is a derivation (or parse) tree for G if:
  1. Every vertex has a label, which is a symbol of $V \cup T \cup \{\epsilon\}$.
  2. The label of the root is S(start symbol).
  3. If a vertex is interior and has label A, then A must be in V.
  4. If n has label A and vertices $n_1, n_2, n_3, ..., n_k$ are the sons of vertex n, in order from the left, with labels $X_1, X_2, ..., X_k$, respectively, then $A \rightarrow X_1X_2.......X_k$ must be a production in P.
  5. If vertex n has label ϵ, then n is a leaf and is the only son of its father.

Derivation Trees (or) Parse tree:

Consider the grammar $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of

$$\begin{aligned} S &\rightarrow aAS|a \\ A &\rightarrow SbA|SS|ba \end{aligned}$$

Construct a derivation tree for the string "aabbaa"

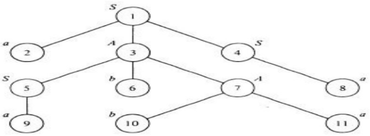


Figure: Derivation tree

$S \implies aAS \implies aSbAS \implies aabAS \implies aabbaS \implies aabbaa.$

Yield of a Tree: The final string obtained by concatenating the labels of the leaves of the tree from left to right, ignoring the Nulls.(eg:-*aabbaa*.)

Ambiguity in context free grammars:

- A context-free grammar G is said to be ambiguous if it has two parse trees for some word.

(or)

- A word which has more than one leftmost derivation or more than one rightmost derivation is said to be ambiguous.

*Note: A CFL for which every CFG is ambiguous is said to be an inherently ambiguous CFL

Example:

- $G = (\{S\}, \{a, b, +, *\}, P.S)$, where P consists of $S \rightarrow S + S|S * S|a|b$ We have two derivation trees for $a + a * b$

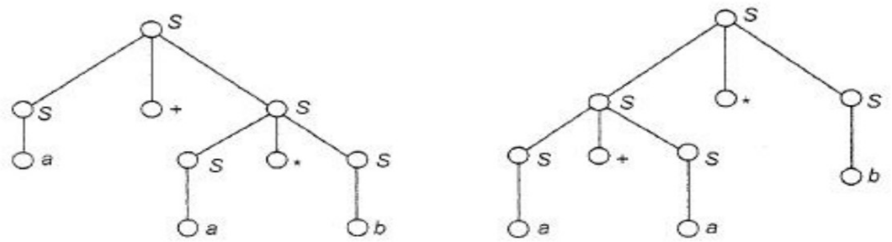


Figure: Two derivation trees for $a + a * b$

Simplified Context free grammar

Reduction of context free grammar

1. We must eliminate useless symbol
   - Eliminate non generating symbols
   - Eliminate non reachable symbols
2. We must eliminate ϵ-production
3. We must eliminate Unit production

Eliminating useless symbol

- Those variables or terminals that do not appears in any derivation of a terminal string from the start symbol
- A symbol γ in CFG is usefull if and ony if
  1. $\gamma \stackrel{*}{\Rightarrow} w$ where $w \in L(G)$ and $w \in T^*$ that is γ leads to a string of terminals. Here γ is said to be generating.
  2. If there is a derivation $S \stackrel{*}{\Rightarrow} \alpha\gamma\beta \stackrel{*}{\Rightarrow} w$ and $w \in L(G)$, for some α and β then γ is said to be reachable.
- So first eliminate symbols that are not generating and then eliminate those symbols are not reachable.

Useful symbol

- A symbol X in a CFG $G = \{V, T, P, S\}$ is called useful if there exist a derviation of a terminal string from S where X appears somewhere, else it is called useless.
    - A symbol X is called generating if some terminal string can be dervied from X.
    - A symbol X is called reachable if it can be reached from S.

Example:

$$\begin{aligned} S &\rightarrow AB/a \\ A &\rightarrow b \end{aligned}$$

Solution:

- Identify nongenerating Symbol
- $S \rightarrow AB/a$ here B is nongenerating.
- The CFG become

$$\begin{aligned} S &\rightarrow a \\ A &\rightarrow b \end{aligned}$$

- A is non reachable
- The CFG become

$$S \rightarrow a$$

Simplified Context free grammar

Eliminate ϵ-production

- ϵ-production are those productions that are of the form $X \rightarrow \epsilon$ these are also called null production
- A variable that derive ϵ ($A \stackrel{*}{\Rightarrow} \epsilon$) then we say A is nullable
- IN CFG, if there is ϵ- production we can remove it without changing the meaning of the grammar.
  - If $A \rightarrow \epsilon$ is a production to be eliminated then we look all production whose right side contain A
  - And replace each occurence of A in each of these productions to obtain the non ϵ-Productions.
  - Now these resultant non ϵ -Production must be added to the grammar to keep the language generated the same.

Simplified Context free grammar

- **Example:**

$$\begin{aligned} S &\rightarrow aA \\ A &\rightarrow b|\epsilon \end{aligned}$$

- $A \rightarrow \epsilon$ is the ϵ-production
- Only one production $S \rightarrow aA$ whose right side contain A , so replace A by ϵ
- we get $S \rightarrow a$
- add this new production to keep the language generated by this grammar same

$$\begin{aligned} S &\rightarrow aA \\ S &\rightarrow a \\ A &\rightarrow b \end{aligned}$$

Eliminate Unit production

- A production i of the form $A \rightarrow B$ is called unit production
- It increase the cost of derivation in a grammar
- If there exist unit production in the grammar
  - Select a unit production $A \rightarrow B$ such that there exist a production $B \rightarrow \alpha$ where α is a terminal
    - For every nonunitproduction $B \rightarrow \alpha$
    - Add production $A \rightarrow \alpha$ to the grammar
    - Eliminate $A \rightarrow B$ from the grammar

Example:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow C|b \\ C &\rightarrow D \\ D &\rightarrow E \\ E &\rightarrow a \end{aligned}$$

Solution

Unit Productions are

$$\begin{aligned} B &\rightarrow C \\ C &\rightarrow D \\ D &\rightarrow E \end{aligned}$$

Simplified Context free grammar

- Remove unit production $B \rightarrow C$ if there exists a production whose left side has C and right side contain a terminal but there is no such production in G,
- Similar things hold for production $C \rightarrow D$.
- Now we try to remove Unit production $D \rightarrow E$, because there is a production $E \rightarrow a$
- Eliminate $D \rightarrow E$ and introduce $D \rightarrow a$
- Grammar becomes

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow C|b \\ C &\rightarrow D \\ D &\rightarrow a \\ E &\rightarrow a \end{aligned}$$

Simplified Context free grammar

- Now we can remove $C \to D$ by using $D \to a$

$$\begin{aligned} S &\to AB \\ A &\to a \\ B &\to C|b \\ C &\to a \\ D &\to a \\ E &\to a \end{aligned}$$

Similarly remove $B \to C$ by using $C \to a$

$$\begin{aligned} S &\to AB \\ A &\to a \\ B &\to a|b \\ C &\to a \\ D &\to a \\ E &\to a \end{aligned}$$

Simplified Context free grammar

- $C \rightarrow a, D \rightarrow a, E \rightarrow a$ are useless symbols so the CFG become

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow a \\
B &\rightarrow a|b
\end{aligned}
$$

Chomsky Normal Form(CNF)

- Any context-free language without ϵ is generated by a grammar in which all productions are of the form $A \rightarrow BC$ or$A \rightarrow a$.
- Here, A, B, and C, are variables and a is a terminal.

Converting CFG into CNF

1. Arrange that all bodies of length 2 or more consist only of variable
2. Break bodies of length ≥ 3 into cascade of productions each with a body consisting of two variables.

Chomsky Normal Form(CNF)

Steps for converting CFG into CNF

Step 2: Simplify the grammar.

a. Eliminate ϵ –productions
b. Eliminate unit productions
c. Eliminate Useless symbols.

Chomsky Normal Form(CNF)

Step 3: Eliminate terminals from RHS if they exist with other terminals or non-terminals.

- Consider a production in P,of the form $A \rightarrow X_1X_2X_3.....X_m$ where $m \geq 2$.If X_i is a terminal,
- Introduce a new variable Ca and a production $C_a \rightarrow a$.
- Then replace X_i by C_a.

Step 4:

- Consider a production $A \rightarrow B_1B_2B_3.....B_m$ where $m \geq 3$,
- Create new variables $D_1, D_2, ...D_{m-2}$ and replace $A \rightarrow B_1B_2B_3...B_m$ by the set of productions $\{A \rightarrow B_1D_1, D_1 \rightarrow B_2D_2, ...D_{m-3} \rightarrow B_m - 2D_{m-2}, D_{m-2} \rightarrow B_{m-1}B_m\}$

Chomsky Normal Form(CNF)

Example: Consider the grammar $(\{S, A, B\}, \{a, b\}, P, S)$ that has the productions:

$$\begin{aligned} S &\rightarrow bA|aB \\ A &\rightarrow bAA|aS|a \\ B &\rightarrow aBB|bS|b \end{aligned}$$

Sln:

Step 1: Simplify the grammar.

- The given grammar does not contain ϵ –productions, unit productions and useless symbols.
- It is in optimized form.

Chomsky Normal Form(CNF)

Step 2:

- The only productions already in proper form are $A \to a$ and $B \to b$.
- So we may begin by replacing terminals on the right by variables, except in the case of the productions $A \to a$ and $B \to b$.
- $S \to bA$ is replaced by $S \to C_bA$ and $C_b \to b$.
- Similarly, $A \to aS$ is replaced by $A \to C_aS$ and $C_a \to a$; $A \to bAA$ is replaced by $A \to C_bAA$; $S \to aB$ is replaced by $S \to C_aB$; $B \to bS$ is replaced by $B \to C_bS$, and $B \to aBB$ is replaced by $B \to C_aBB$.

Step 3:

- In the next stage, the production $A \to C_bAA$ is replaced by $A \to C_bD1$ and $D1 \to AA$, and the production $B \to C_aBB$ is replaced by $B \to C_aD2$ and $D2 \to BB$.

Chomsky Normal Form(CNF)

The productions for the grammar in CNF are :

$$
\begin{aligned}
S &\rightarrow C_bA|C_aB \\
A &\rightarrow C_aS|C_bD_1|a \\
B &\rightarrow C_bS|C_aD_2|b \\
C_b &\rightarrow b \\
D1 &\rightarrow AA \\
D2 &\rightarrow BB \\
Ca &\rightarrow a \\
Cb &\rightarrow b
\end{aligned}
$$

Normal Form

Left Recursion

- A grammar which is of the form $A \rightarrow A\alpha|\beta$
- To avoid left recursion

$$\begin{aligned} A &\rightarrow \beta A' \\ A' &\rightarrow \alpha A' \\ A' &\rightarrow \epsilon \end{aligned}$$

- After removing ϵ production the grammar become

$$\begin{aligned} A &\rightarrow \beta|\beta A' \\ A' &\rightarrow \alpha|\alpha A' \end{aligned}$$

Greibach Normal Form(GNF)

- Every context free language L without ϵ can be generated by a grammar for which every production is of the form $A \rightarrow a\alpha$ or $A \rightarrow a$
- Where A is a variable , a is a terminal and α is a string of variables.

To convert given grammar into GNF following Two lemma's are considered

Lemma1:

- Define an A-production to be a production with variable A on the left.
- Let $G = (V, T, P, S)$ be a CFG. Let $A \rightarrow \alpha_1 B \alpha_2$ be a production in P and $B \rightarrow \beta_1|\beta_2|....|\beta_r$ be the set of all B-productions.
- Let $G1 = (V, T, P1, S)$ be obtained from G by deleting the production $A \rightarrow \alpha_1 B \alpha_2$ from P and adding the productions $A \rightarrow \alpha_1\beta_1\alpha_2|\alpha_1\beta_2\alpha_2|...|\alpha_1\beta_r\alpha_2$.
- Then $L(G) = L(G1)$.

Greibach Normal Form(GNF)

Lemma2:

- Let $G = (V, T, P, S)$ be a CFG. Let $A \rightarrow A\alpha_1|A\alpha_2|...|A\alpha_r$ be the set of A-productions for which A is the leftmost symbol of the right-hand side.
- Let $A \rightarrow \beta_1|\beta_2|...|\beta_s$ be the remaining A-productions.
- Let $G1 = (V \cup \{B\}, T, P1, S)$ be the CFG formed by adding the variable B to V and replacing all the A-productions by the productions:

$$\left.\begin{aligned} A \rightarrow &\quad \beta_i \\ A \rightarrow &\quad \beta_i B \end{aligned}\right\} 1 \leq i \leq s \qquad\qquad \left.\begin{aligned} B \rightarrow &\quad \alpha_i \\ B \rightarrow &\quad \alpha_i B \end{aligned}\right\} 1 \leq i \leq r$$

- Then $L(G1) = L(G)$.

Algorithm for converting a CFG into GNF

1. Remove Unit productions, useless symbols and ϵ-production, if any.
2. For every terminal symbol introduce a new non terminal.
   - $A \rightarrow Ba$ become $A \rightarrow BC_a$, $C_a \rightarrow a$
3. Introduce an order among nonterminals by renaming them.
4. Using substitutions, rewrite productions (except Left Recursive) if required, to ensure that all productions of the form $X_i \rightarrow X_j\alpha$ satisfy the condition $i < j$
5. Remove Left Recursions, if any.
6. Remove unit productions,useless symbol if any added in step 4.
7. Obtain the CFG in GNF by applying substitutions.

Greibach Normal Form(GNF)

Example: Convert the following grammar into GNF

$$\begin{aligned} S &\rightarrow AB|CC \\ C &\rightarrow b|SC \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Sln:

- The given CFG is in CNF
- Renaming the nonterminals in G with index values
  - Replace, S with X_1,A with X_2 ,B with X_3,C with X_4
- Equivalent CFG G'

$$\begin{aligned} X_1 &\rightarrow X_2X_3|X_4X_4 \\ X_4 &\rightarrow b|X_1X_4 \\ X_2 &\rightarrow a \\ X_3 &\rightarrow b \end{aligned}$$

Example cont..

- The production $X_4 \rightarrow X_1X_4$ does not satisfy our requirement we need $i < j$
- Now, substitute the value of X_1 in X_4 we will get $X_4 \rightarrow X_2X_3X_4 | X_4X_4X_4$
- Substitute $X_2 \rightarrow a$ into $X_4 \rightarrow X_2X_3X_4$ then $X_4 \rightarrow aX_3X_4$
- The production $X_4 \rightarrow X_4X_4X_4$ is left recursive remove the leftrecursive
- the grammar become

$$
\begin{aligned}
X_1 &\rightarrow X_2X_3 | X_4X_4 \\
X_4 &\rightarrow b | aX_3X_4 | bX_5 | aX_3X_4X_5 \\
X_2 &\rightarrow a \\
X_3 &\rightarrow b \\
X_5 &\rightarrow X_4X_4 | X_4X_4X_5
\end{aligned}
$$

Example cont..

- Againg perform substitution then the grammar become

$$\begin{aligned}
X_1 &\rightarrow aX_3|X_4X_4 \\
X_4 &\rightarrow b|aX_3X_4|bX_5|aX_3X_4X_5 \\
X_2 &\rightarrow a \\
X_3 &\rightarrow b \\
X_5 &\rightarrow X_4X_4|X_4X_4X_5
\end{aligned}$$

- Againg substitute the value of X_4 then the grammar become

$$\begin{aligned}
X_1 &\rightarrow aX_3|bX_4|aX_3X_4X_4|bX_5X_4|aX_3X_4X_5X_4 \\
X_4 &\rightarrow b|aX_3X_4|bX_5|aX_3X_4X_5 \\
X_2 &\rightarrow a \\
X_3 &\rightarrow b \\
X_5 &\rightarrow X_4X_4|X_4X_4X_5
\end{aligned}$$

Example cont..

- Againg substitute the value of X_4 then the grammar become

$$
\begin{aligned}
X_1 &\rightarrow aX_3|bX_4|aX_3X_4X_4|bX_5X_4|aX_3X_4X_5X_4 \\
X_4 &\rightarrow b|aX_3X_4|bX_5|aX_3X_4X_5 \\
X_2 &\rightarrow a \\
X_3 &\rightarrow b \\
X_5 &\rightarrow bX_4|aX_3X_4X_4|bX_5X_4|aX_3X_4X_5X_4| \\
&\quad\ bX_4X_3|aX_3X_4X_4X_5|bX_3X_4X_5|aX_3X_4X_5X_4X_5
\end{aligned}
$$