→ Understanding the language defined by Grammars

$G = \{\{S\}, \{a\}, \{S \to SS\}, S\}$

$L(G) = \phi$.

→ $G = \{\{S, C\}, \{a, b\}, P, S\}$ where P is given by.

$S \to aCa$

$C \to aCa|b$.

$L = \{$

aCa.
aba.,
aCa.
aaCaa
$a^n b a^n$.

$L = \{a^n b a^n : n \geq 1\}$.

→ $G = \{\{S\}, \{0, 1\}, P, \{\}\}$

$P = \to S \to 0S1|\epsilon$.

$L = \{0^n 1^n, n \geq 0\}$

ϵ, 01, 0011.
0S1
00S11
0011

Left and Right most derivations

The left most non-terminal in a working string is the first non-terminal that we encounter when we scan the string from left to right.

Ex: bbabXbaY. S bXbY, the left most NT is X.

Def: If a word w is generated by a CFG by a certain derivation and at each step in the derivation a rule of production is applied to the left most [illegible] in the working string,

this derivation is called left most derivation.

Similarly, replacing the right most variable first at every step gives the Right most derivation.

Consider the CFG $(\{S, X\}, \{a, b\}, P, S)$

$S \rightarrow baXa \mid Sab.$

$X \rightarrow Xab \mid aa.$

Given $w = baaaababaab.$

Find LMD and RMD.

Sol: LMD $S \rightarrow baXas.$

$baXabas \quad X \rightarrow Xab.$

$baXababas \quad X \rightarrow Xab$

$baaaababas \quad X \rightarrow aa.$

$baaaababaab \quad S \rightarrow ab.$ ✓

RMD :- $S \rightarrow baXas.$

$baXaab \quad \because S \rightarrow ab$

$baXababaab \quad X \rightarrow Xab$

$baXababaab \quad X \rightarrow Xab$

$baaababaab \quad X \rightarrow aa.$ ✓

~~Derivation Tree~~

Application of CFG

→ Grammars are useful in specifying the syntax of programming languages. They are mainly used in the design of programming languages

→ They are also used in NLP.

→ Tamil poetry called Venpa is described by a CFG.

→ CFGs are used in speech recognition and also in processing spoken words.

Applications of RE & FA

lexical analyzers and text editors are 2 applications

↓

The tokens of programming lang can be expressed using RE. The lexical analyzer scan the i/p program and seperate the tokens.

Ex: identifier can be expressed as a RE.

(letter)(letter + digit)* where letter is [A-Z a-z] and digit is [0-9]

If any thing in the source language matches with the RE given above, it is recognized as an identifier.

RE can be used to identify the tokens in a lang

Text Editors

These are the programs used for processing the text.

Ex: UNIX text editors use the RE for substituting the strings such as $s/bbb^*/b/$.

This indicates to replace the multiple blanks spaces with single blank.

In Unix text editors, any RE is converted to NFA with ϵ. This NFA can be simulated directly.

Derivation Tree

The derivation process can be shown pictorially as a tree to illustrate how a word is derived from a CFG.

These trees are called syntax trees, parse trees or derivation trees.

$G = (V, T, P, S)$.

→ root is a starting symbol.

→ Each node is labeled by either a variable or terminal of G.

→ Each interior node is marked by a variable in V.

→ if an interior node is labelled A, and its childrener are labelled $X_1, X_2 \cdots X_k$ From left to right $A \to X_1 X_2 \cdots X_k$ is a

Production in P.

CF Grammar

$T \rightarrow a|b$

$NT \rightarrow S, A.$

$P \Rightarrow S \rightarrow AAA \mid AA$

$A \rightarrow AA \mid aA \mid Ab \mid a|b$ → (baaba).

Sentential form.

Ambiguous Grammar

A CFG is ambiguous if there exist more than one parse tree. or equivalently, if there exist more than one LMD and thus there exist more than one RMD for atleast one word in CFL

Ambiguous. → There exists more than one LMD or RMD for a string.

→ LMD & RMD represents different parse trees.

→ More than one parse tree for a string.

Unambiguous grammar → Unique LMD | RMD.

→ LMD & RMD represents same parse tree.

→ unique parse tree.

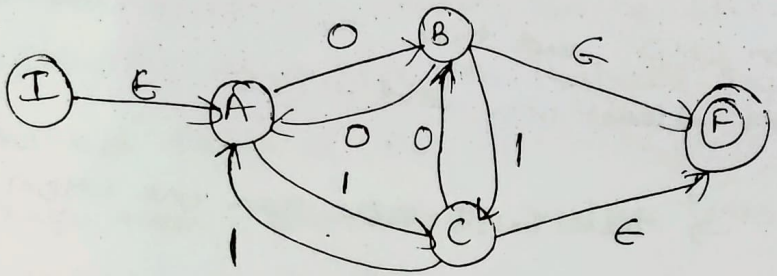
→ A sentential form in the start symbol S of a grammar G any string in $(V \cup T)^*$ that can be derived from S.

$(\{S, B\}, \{a, b\}, S, \{S \to aS, S \to B, B \to bB, B \to \epsilon\})$

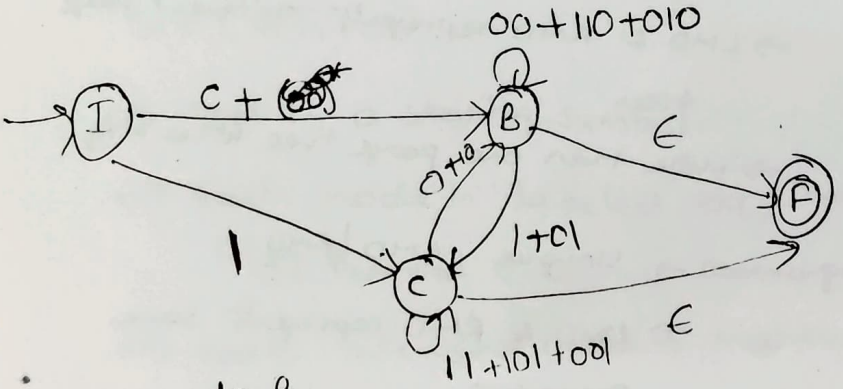
A derivation using this grammar

$S \Rightarrow aS$
$\Rightarrow aB$
$\Rightarrow abB$
$\Rightarrow abbB$
$\Rightarrow abb$

In this $\{S, aS, aB, abB, abbB, abb\}$ is a sentential form



Eliminate A

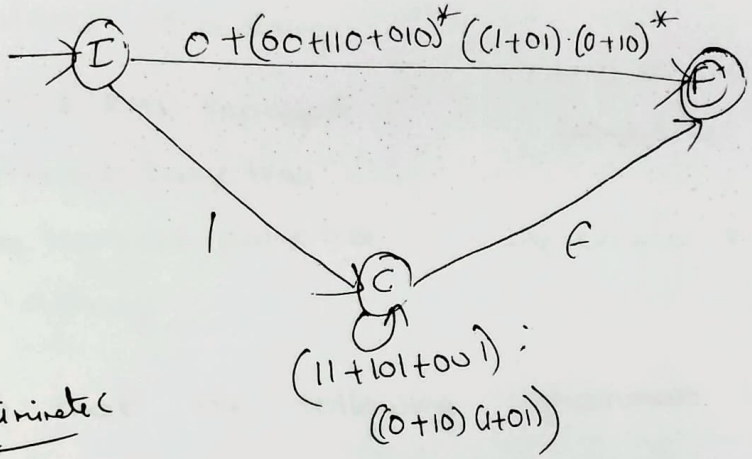


Eliminate B

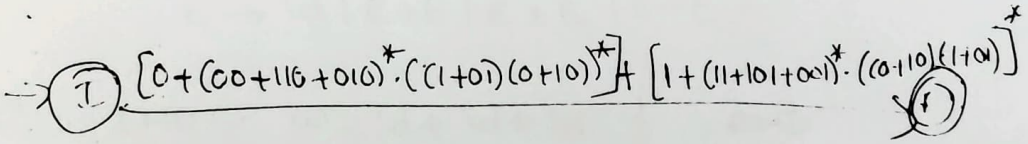
6/2/18

C1, C2, C3, C4, [illegible], C8, C9, D0, D1, D2, D4, D5, D6
D8, [illegible] E0, E1, E2, E6, E7, E9, F0, F1, [illegible], f4, f6
f9, G1, G2, G3, G4, [illegible], G7, [illegible] H0, H2, H3, H4,
H6, H7, H8, [illegible] T- 86, Le- 27, 28, 29, 30, 31,
32, 33, [illegible]

Eliminate B



eliminate C

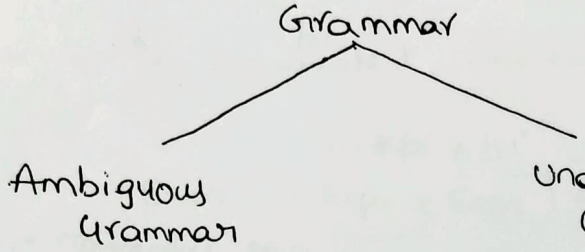


Ambiguous Grammar:-

A CFG is ambiguous if there exist more than one parse tree or equivalently, if there exist more than one leftmost derivation and thus there exist more than one rightmost derivation, for atleast one word in its CFL.

S → iCtS | iCtSeS | a
C → b.

i b t i b t a e a



→ there Exists more than ONE LMD|RMD for a string
→ LMD & RMD represents different parse trees
→ More than one parse tree for a string

→ Unique LMD|RMD.
→ LMD & RMD represents same Parse tree.
→ Unique parse tree.

→ Show that the following grammar is ambiguous.

$$E \rightarrow id \mid E+E \mid E*E \mid E-E.$$

Sol:

LMD: w = 'id + id * id.

$E \rightarrow \underline{E} + E$
$\rightarrow id + \underline{E}$
$\rightarrow 'id + \underline{E} * E$
$\rightarrow id + 'id * E$
$\rightarrow id + id * id$

RMD.

$E \rightarrow E * \underline{E}$
$\rightarrow E * id$
$\rightarrow E + \underline{E} * id$
$\rightarrow E + id * id$
$\rightarrow id + id * id$

removing

Parse trees

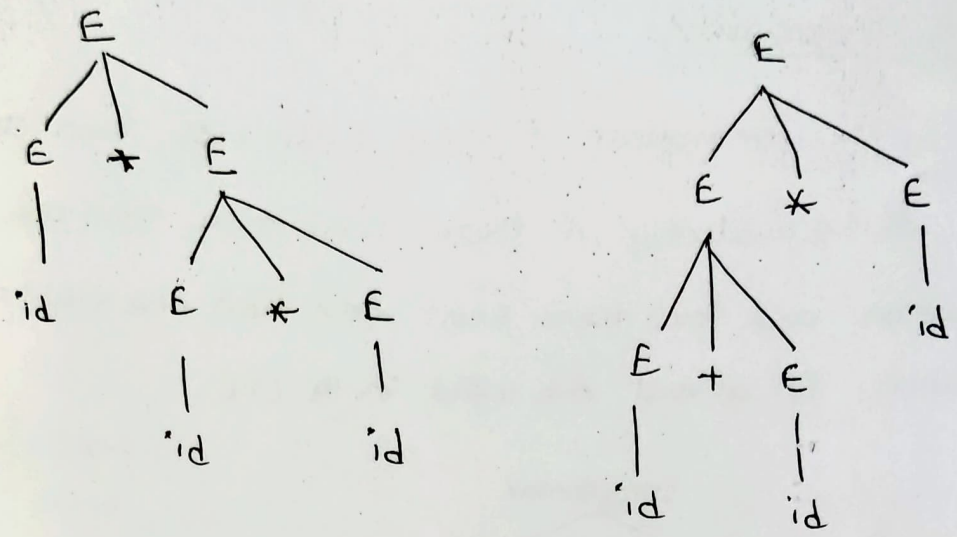


fig:- Parse tree for "'id + id * id"

As there are more than one parse tree, Grammar is ambiguous.

→ Consider Grammar G with.

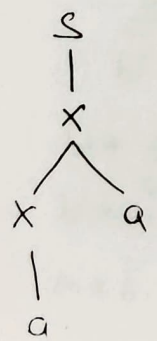
$S \to as \mid asb \mid X$

$X \to Xa \mid ba$.

Show that G is ambiguous.

Sol: $w = aa$.

$S \Rightarrow X$
$\Rightarrow Xa$.
$\to aa$



$S \Rightarrow as$
$\Rightarrow ax$
$\Rightarrow aa$.

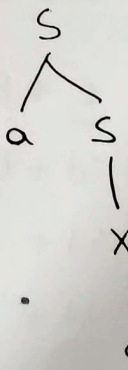


fig :- Parse trees for string w = aa

Removing Ambiguity

There is no algorithm that straightway converts an ambiguous grammar to Equivalent Unambiguous grammar. But on analysing the grammar, if we identify what is missing in the grammar and why it is ambiguous, then we can write Equivalent Unambiguous grammars.

for Ex, $Expr \rightarrow Expr + Expr \mid Expr * Expr \mid id$.

After analysing the above grammar,

1. Precedence of operators is not taken care of; hence you can derive the string either replacing Expr by Expr + Expr or by Expr * Expr.
2. Associative property is also not taken care of. if we take a string id+id+id, we can replace Expr → Expr + Expr Either the left Expr or right Expr.

for Ex, in the given grammar 'id' has the highest precedence, then is '*' and least precedence is for '+'.

So, do not define all of them at the same level.

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow id$$

Push Down Automata.

The concept PDA is useful in designing Parsers of syntax analysers. A Parser verifies the syntax of the text. Parsing is a part of the compilation process.

PDA consists of three components

1. Input tape
2. finite control
3. stack structure.

An input tape consists of a linear configuration of cells Each of which contains a character from the input alphabet. This tape can be moved one cell at a time to the left The control unit has some pointer (head) which points the current symbol that is to be read.

The stack is also sequential structure. The head is positioned over the current stack element can read and write special stack characters from that position.

The control unit contains both the tape head and stack head and finds itself at any moment in a particular state.

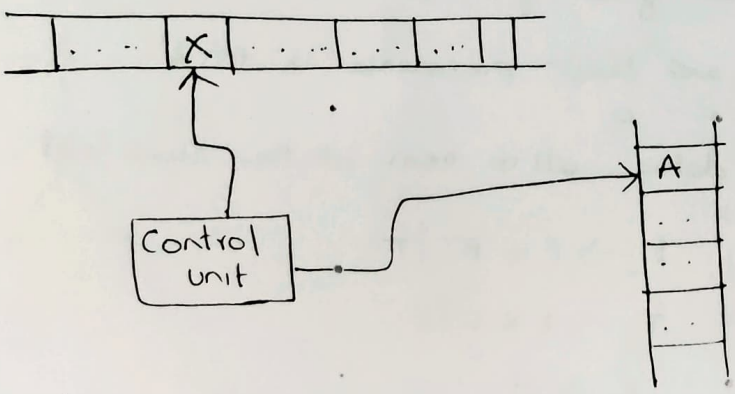


fig: Conceptual model of PDA

Definition :- A finite state Pushdown Automata is a 7 tuple

$M = (Q, \Sigma, \delta, T, q_0, z_0, F)$ where.

$Q \rightarrow$ a finite set of states

$\Sigma \rightarrow$ a finite set of input alphabet

$T \rightarrow$ a finite set of stack alphabet

$q_0 \rightarrow$ the starting state $q_0 \in Q$.

$F \subseteq Q \rightarrow$ a set of final states

$\delta \rightarrow$ transition function from

$$Q \times (\Sigma \cup \{\varepsilon\}) \times (T) \rightarrow Q \times T^*$$

$z_0 \rightarrow$ is the initial stack-symbol. $z_0 \in T$.

→ ① $\delta(q, a, z_0) = (q, az_0)$ indicates that in the state q on seeing a, a is pushed on to stack. There is no change of state.

② $\delta(q, a, z_0) = (q, \varepsilon)$ indicates that in the state q on seeing a the current top symbol z_0, is deleted from the stack.

③ $\delta(q, a, z_0) = (q_1, az_0)$ indicates that a is pushed on to the stack and the state is changed to q_1.

Graphical Representation of PDA:-

Let $M = (Q, \Sigma, T, \delta, q_0, z_0, F)$ be a PDA where $Q = \{p, q\}$, $\Sigma = \{a, b, c\}$, $T = \{a, b\}$, $q_0 = q$, $F = \{p\}$, and δ is given by the following equations:

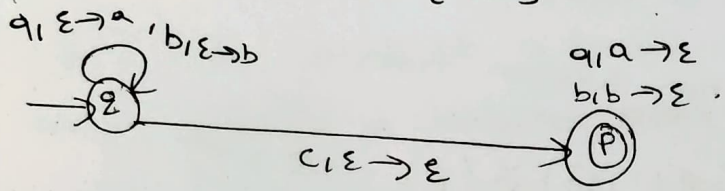
$$\delta(q, a, \varepsilon) = \{(q, a)\}$$

$$\delta(q, b, \varepsilon) = \{(q, b)\}$$

$$\delta(q, c, \varepsilon) = \{(p, \varepsilon)\}$$

$$\delta(p, a, a) = \{(p, \varepsilon)\}$$

$$\delta(p, b, b) = \{(p, \varepsilon)\}$$



Instantaneous Description of PDA

During processing, the PDA moves from one configuration to another configuration.

At any given instance, the configuration of PDA is expressed by the state or finite control, the content of stack and the input.

The configuration is expressed as a triple (q, x, γ) where

1. q is the state
2. x is the input string to be processed.
3. γ is the content of the stack where the leftmost symbol corresponds to top of stack and the rightmost is the bottom element

When string abab_bab is processed, the instantaneous description is shown below

$(q_1, ababcbab, \varepsilon) \Rightarrow (q_1, babcbab, a) \Rightarrow (q_1, abcbab, ba)$

$\Rightarrow (q_1, bcbab, aba) \Rightarrow (q_1, cbab, baba) \Rightarrow (p_1, bab, baba)$

$\Rightarrow (p, ab, aba)$

$\Rightarrow (p, b, ba) \Rightarrow (p, \varepsilon, a)$.

At this point, the i/p string is exhausted and the computation stops. we cannot accept the original string although we are in 'accept' state because the stack is not empty.

→ Design PDA that accepts $L = \{0^n 1^{2n} \mid n \geq 1\}$

Here, we have to match each '0' with two 1's.

Let q_0 be initial state, q_f be final state and z_0 be the bottom of the stack.

$\delta(q_0, 0, z_0) = (q_0, 0z_0)$

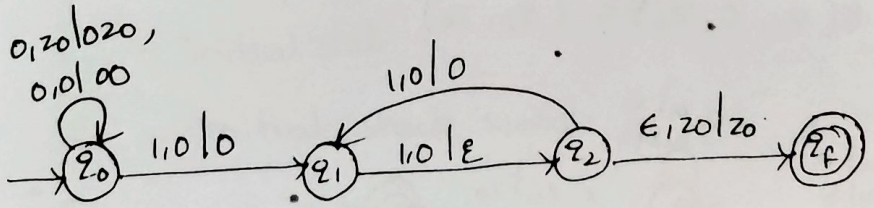
$\delta(q_0, 0, 0) = (q_0, 00)$

$\delta(q_0, 1, 0) = (q_1, 0)$

$\delta(q_1, 1, 0) = (q_2, \varepsilon)$

$\delta(q_2, 1, 0) = (q_1, 0)$

$\delta(q_2, \epsilon, z_0) = (q_f, z_0)$

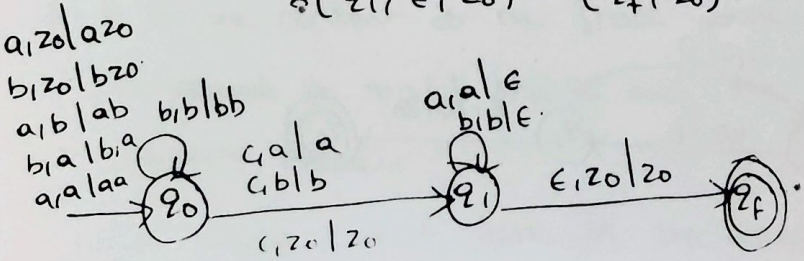


3. Design PDA which accepts $L = \{ w c w^r \mid w \in (a+b)^* \}$

Read the string 'w' and push it on to the stack until it encounters c. After that, read each symbol, if it matches with the top of the stack, POP off the symbol.

Let q_0 be initial state, q_f be final state and z_0 be the bottom / initial stack symbol.

$$\delta(q_0, a, z_0) = (q_0, az_0)$$
$$\delta(q_0, b, z_0) = (q_0, bz_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, c, z_0) = (q_1, z_0)$$
$$\delta(q_0, c, a) = (q_1, a)$$
$$\delta(q_0, c, b) = (q_1, b)$$
$$\delta(q_1, a, a) = (q_1, \epsilon)$$
$$\delta(q_1, b, b) = (q_1, \epsilon)$$
$$\delta(q_1, \epsilon, z_0) = (q_f, z_0).$$



Equivalence of Acceptance of final state and Empty Stack:-

Theorem 1: If L is $L(M_1)$ for some PDA M_1 that accepts using final state, then L is $L(M_2)$ for some PDA M_2 that accepts using the empty stack.

Proof:- Let $M_1 = (Q_1, \Sigma_1, \delta_1, \Gamma_1, q_{01}, Z_1, q_{f1})$ where PDA M_1 accepts the strings by entering into final state i.e.,

$$(q_{01}, x_1, z_1) \overset{*}{\Rightarrow} (q_{f1}, \varepsilon, \gamma)$$

we need to construct PDA M_2 such that it accepts the strings of L.

∴ we need to find

$$M_2 = \{Q_2, \Sigma_2, \Gamma_2, \delta_2, q_2, Z_2, \phi\} \text{ such that}$$

$$L(M_2) = L.$$

let $Q_2 = Q_1 \cup \{q_2, q_e\}$

$\Sigma_2 = \Sigma_1 \cup \{\varepsilon\}$

$\Gamma_2 = \Gamma_1 \cup \{Z_2\}$

q_2 – initial state

Z_2 – initial stack symbol.

δ_2 is as follows.

$\delta_2(q_2, \varepsilon, Z_2) = \delta_2(q_{01}, Z_1 Z_2)$

Now simulates moves of M_2 on M_2.

$$\delta_2(q, a, X) = \delta_1(q, a, X).$$

for all $q \in Q_1$ $a \in \Sigma_1$ & $X \in T_1$.

If M_1 Enters in to final state, then moves in M_2 are defined to move to q_e state.

$$\delta_2(q_{f1}, \varepsilon, X) = (q_e, X)$$

$$\delta_2(q_e, \varepsilon, X) = (q_e, \varepsilon)$$

for all $X \in T$.

To show $x \in L(M_2)$

$$\delta_2(q_2, x, z_2) \Rightarrow^* (q_1, x, z_1 z_2) \overset{*}{\Rightarrow} (q_{f1}, \varepsilon, \gamma) \Rightarrow (q_e, \varepsilon, \gamma) \Rightarrow (q_e, \varepsilon, \varepsilon).$$

<u>Theorem:2</u> If L is $L(M_3)$ for some PDA M_3 which accepts using Empty stack, then L is $L(M_4)$ for some PDA M_4 that accepts using final state.

<u>Proof:</u> Let $M_3 = \{Q_3, \Sigma_3, T_3, \delta_3, q_3, z_3, \phi\}$, where the PDA M_3 accepts the strings by Emptying the stack i.e.,

$$(q_3, x, z_3) \overset{*}{\Rightarrow} (q, \varepsilon, \varepsilon) \quad \forall\, q \in Q_3.$$

we need to construct PDA M_4 such that it accepts string of L.

we need to find

$M_4 = \{Q_4, \Sigma_4, \Gamma_4, \delta_4, q_4, Z_4, q_{f4}\}$ such that $L(M_4) = L$

Let $Q_4 = Q_3 \cup \{q_4, q_{f4}\}$.

$\Sigma_4 = \Sigma_3 \cup \{\varepsilon\}$,

$\Gamma_4 = \Gamma_3 \cup \{Z_4\}$.

q_4 — initial state,

Z_4 — initial stack symbol,

q_{f4} — final state,

δ_4 is defined as follows.

Add transition in M_4 to move to initial state of M_3, to process the string

$\delta_4(q_4, \varepsilon, Z_4) = (q_3, Z_3 Z_4)$.

Simulate all moves of M_3 on M_2

$\delta_4(q, a, X) = \delta_3(q, a, X)$ for all $q \in Q_3$, $a \in \Sigma_3$, $X \in \Gamma_3$

Even if M_3 emptied the stack content, there would be Z_4 on the stack. Now move to final state.

$\delta_4(q, \varepsilon, Z_4) = (q_{f4}, \varepsilon Z_4)$ for all $q \in Q_3$.

To show $x \in L(M_4)$

$(q_4, x, Z_4) \Rightarrow (q_3, x, Z_3 Z_4) \overset{*}{\Rightarrow} (q, \varepsilon, Z_4) \Rightarrow (q_{f4}, \varepsilon, Z_4)$.

Equivalence of PDA and Context free grammars.

Constructing PDA for given CFG.

Let G be given as (V, T, P, S) be CFG in the Greibach normal form. It is required to construct $M = (\{q\}, T, V, \delta, q, S, \phi)$ where $\delta(q, a, A)$ contains (q, γ) whenever $A \to a\gamma$ is in P.

The PDA M simulates leftmost derivations of G. since G is in GNF, each sentential form in a leftmost derivation consists of terminals x followed by a string of variables α.

M stores the suffix α of the left sentential form on its stack after processing the prefix x. formally we show that

$S \Rightarrow x\alpha$ by a leftmost derivation if and only if

$$(q, x, S) \overset{*}{\underset{M}{\vdash}} (q, \varepsilon, \alpha)$$

→ Construct a PDA equivalent to the following grammar.

$S \to aAA$, $A \to aS \mid bS \mid a$.

Sol:- The grammar is in GNF. Hence we can apply the rules as follows.

$S \to aAA$ corresponds $\delta(q, a, S) = (q, AA)$

$A \to aS$ " $\delta(q, a, A) = (q, S)$

$A \to bS$ " $\delta(q, b, A) = (q, S)$

$A \to a$ " $\delta(q, a, A) = (q, \varepsilon)$

$S \rightarrow ABa \mid BC$

$A \rightarrow aC \mid BCC$

$C \rightarrow a$

$B \rightarrow bCC$

→ Eliminate useless symbols from grammar.

$S \rightarrow aAa$.

$A \rightarrow bBB$

$B \rightarrow ab$

$C \rightarrow ab$.

Sol:- C is unreachable from S.

$S \rightarrow aAa$

$A \rightarrow bBB$

$B \rightarrow ab$. is the final grammar.

② Elimination of ϵ - productions:

Eliminate the null production from the following grammar:

$S \rightarrow ABaC$

$A \rightarrow BC$

$B \rightarrow b \mid \epsilon$

$C \rightarrow D \mid \epsilon$

$D \rightarrow d$.

Sol:- ϵ - variables are $V_n = \{B, C, A\}$

$S \rightarrow ABaC \mid BaC \mid ABa \mid AaC \mid Aa \mid Ba \mid aC \mid a$.

$A \rightarrow BC \mid B \mid C$

$B \rightarrow b$

$C \rightarrow D$

$D \rightarrow d$

Simplification of Grammars:-

Languages can effectively be represented by CFG. All grammars are not always optimized. That means, a grammar may contain some extra/unnecessary symbols. These will increase the length of the grammar.

Simplification of the grammar, generally includes the following:

1. Elimination of useless symbols.
2. Elimination of ϵ-productions.
3. Elimination of unit productions of the form $A \to B$.

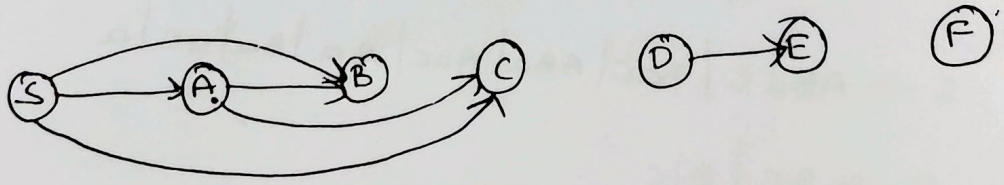
(1) Elimination of Useless symbols.

$S \to ABa \mid BC$, $A \to aC \mid BCC$, $C \to a$, $B \to bCC$,

$D \to E$, $E \to d$, $F \to e$

(1) All variables are found to be useful as each of them derive a terminal symbol.

(2) Elimination of non reachable variable. Draw the dependency graph shown below.



Here D, E, F are non reachable from S.

So, remove these symbols from Grammar.

(3) Elimination of unit productions.

A unit production is a production of the form $A \to B$.

→ Eliminate unit productions in the grammar.

$S \to A \mid bb$.

$A \to B \mid b$.

$B \to S \mid a$.

Eliminating unit production means replace the production with either terminal or $(VUT)^*$.

$S \to A \mid bb \Rightarrow \quad S \to b \mid a \mid bb$

$A \to B \mid b \Rightarrow \quad A \to a \mid b \mid bb$.

$B \to S \mid a \Rightarrow \quad B \to a \mid b \mid bb$.

Design PDA that accepts $L = \{0^n 1^{2n} \mid n \geq 1\}$

Here we have to match each 0s with two 1's. So, read the 0's and push the double zero on the stack. After reading 1' we can change the state and pop the zero from the stack. when the input is read completely, if the stack becomes empty, then it is successful.

Let q_0 be the initial state, q_f be final state and Z_0 be the bottom of the stack.

$$\delta(q_0, 0, Z_0) = (q_0, 00Z_0)$$

$$\delta(q_0, 0, 0) = (q_0, 00)$$

$$\delta(q_0, 1, 0) = (q_1, \epsilon)$$

$$\delta(q_1, 1, 0) = (q_1, \epsilon).$$

$$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0).$$

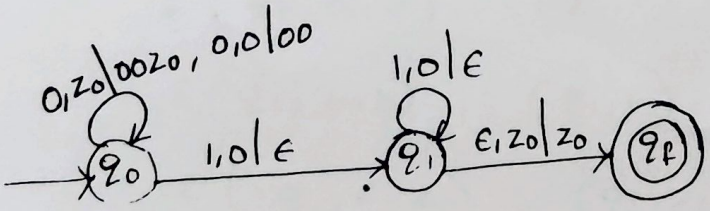


fig: transition diagram.

Design PDA that accepts $L = \{a^3 b^n c^n / n \geq 0\}$

Here, exactly three a's are read initially and push on to the stack. Then read all b's on to the stack and match them with c's by popping off.

(δ)

$\delta(q_0, a, z_0) = (q_1, az_0)$

$\delta(q_1, a, a) = (q_2, a)$

$\delta(q_2, a, a) = (q_3, a)$

$\delta(q_3, \epsilon, a) = (q_f, a)$

$\delta(q_3, b, a) = (q_4, ba)$

$\delta(q_4, b, b) = (q_4, bb)$

$\delta(q_4, c, b) = (q_5, \epsilon)$

$\delta(q_5, c, b) = (q_5, \epsilon)$

$\delta(q_5, \epsilon, a) = (q_f, a)$

$\delta(q_0, a, z_0) = (q_1, z_0)$

$\delta(q_1, a, z_0) = (q_2, z_0)$

$\delta(q_2, a, z_0) = (q_3, z_0)$

$\delta(q_3, \epsilon, z_0) = (q_f, z_0)$

$\delta(q_3, b, z_0) = (q_4, bz_0)$

$\delta(q_4, b, b) = (q_4, bb)$

$\delta(q_4, c, b) = (q_5, \epsilon)$

$\delta(q_5, c, b) = (q_5, \epsilon)$

$\delta(q_5, \epsilon, z_0) = (q_f, z_0)$

Design PDA which accepts $L = \{ w c w^r \mid w \in (a+b)^* \}$

Read the string w and push it on to the stack until it encounters c. After that read each symbol, if it matches with the top of the stack pop off the symbol. when the i/p is read completely, if stack becomes empty then string is accepted.

$$\delta(q_0, a, z_0) = (q_0, a z_0)$$

$$\delta(q_0, b, z_0) = (q_0, b z_0)$$

$$\delta(q_0, a, a) = (q_0, aa)$$

$$\delta(q_0, a, b) = (q_0, ab)$$

$$\delta(q_0, b, b) = (q_0, bb)$$

$$\delta(q_0, b, a) = (q_0, ba)$$

$$\delta(q_0, c, a) = (q_1, a)$$

$$\delta(q_0, c, b) = (q_1, b)$$

$$\delta(q_1, a, a) = (q_1, \epsilon)$$

$$\delta(q_1, b, b) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, z_0) = (q_f, z_0)$$

Equivalence of Acceptance of final state and Empty stack.

Theorem I: If L is $L(M_1)$ for some PDA M_1 that accepts using final state, then L is $L(M_2)$ for some PDA M_2 that accepts using the empty stack.

Proof: Let $M_1 = \{Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, Z_1, q_{f1}\}$ where the PDA M_1 accepts the strings by entering in to final state

i.e.,

$$(q_1, x, Z_1) \Rightarrow (q_{f1}, \epsilon, Y)$$

we need to construct a PDA M_2 such that it accepts the strings of L.

∴ we need to find

$$M_2 = \{Q_2, \Sigma_2, \Gamma_2, \delta_2, q_2, Z_2, \phi\}$$

such that $L(M_2) = L$.

Let $Q_2 = Q_1 \cup \{q_2, q_e\}$

$\Sigma_2 = \Sigma_1 \cup \{\epsilon\}$

$\Gamma_2 = \Gamma_1 \cup \{Z_2\}$

q_2 — initial state

Z_2 — initial stack symbol.

δ_2 is defined as follows

→ Add transition in M_2, to move initial state of M_1 to process the strings.

$$\delta_2(q_2, \epsilon, Z_2) = (q_1, Z_1 Z_2)$$

Simulate the moves of M_1 on M_2:

$$\delta_2(q, a, X) = \delta_1(q, a, X)$$

for all $q \in \{Q_1\} - q_{f1}$, $a \in \Sigma$, and $X \in T_1$.

If M_1 enters into final state, then moves in M_2 are defined to move to q_e state, which will empty the stack contents

$$\delta_2(q_{f1}, \epsilon, X) = (q_e, \epsilon, Y)$$

$$\delta_2(q_e, \epsilon, X) = (q_e, \epsilon, \epsilon)$$

for all $X \in T$.

To show $x \in L(M_2)$

$$(q_2, x_2, Z_2) \vdash (q_1, x_1, Z_1 Z_2) \overset{*}{\Rightarrow} (q_{f1}, \epsilon, Y) \Rightarrow (q_e, \epsilon, Y) \overset{*}{\Rightarrow} (q_e, \epsilon, \epsilon).$$

Theorem 2

If L is $L(M_3)$ for some PDA M_3 which accepts using Empty stack, then L is $L(M_4)$ for some PDA M_4 that accepts using final state.

Proof

Let $M_3 = \{Q_3, \Sigma_3, T_3, \delta_3, q_3, Z_3, \phi\}$ where the PDA M_3 accepts the strings by emptying the stack

$(q_3, x, z_3) \overset{*}{\Rightarrow} (q, \epsilon, \epsilon)$ for $q \in Q_3$

we need to construct PDA M_4 such that it accepts strings of L.

∴ we need to find

$M_4 = \{Q_4, \Sigma_4, \delta_4, q_4, Z_4, q_{f4}\}$ such that $L(M_4) = L$

let $Q_4 = Q_3 \cup \{q_4, q_{f4}\}$

$\Sigma_4 = \Sigma_3 \cup \{\epsilon\}$

$\Gamma_4 = \Gamma_3 \cup \{Z_4\}$

q_4 – initial state

Z_4 – initial stack symbol.

q_{f4} – final state

δ_4 is defined as follows.

Add transition in M_4, to move to initial state of M_3 to process the strings.

$$\delta_4(q_4, \epsilon, Z_4) = \delta_3(q_3, \epsilon, Z_3 Z_4)$$

Simulate the moves of M_3 on M_4.

$\delta_4(q, a, X) = \delta_3(q, a, X)$ for all $q \in \{Q_3\}$, $a \in \Sigma_3$ and $X \in \Gamma_3$

Even if M_3 empties the stack content, there would be Z_4 on the stack. Now move to final state

closure properties of context free languages:

→ context free languages are closed under union, concatenation and Kleene closure.

→ Also under reversal, homomorphisms and inverse homo-morphisms.

→ But not closed under intersection & difference.

Union:-

→ Let L and M be CFL's with Grammars G and H, respectively.

→ Assume G and H have no variables in common.

* Name of the variables do not affect the language

→ Let S_1 and S_2 be the start symbols of G and H

→ Form a new grammar for L ∪ M by combining all the symbols and productions of G and H.

→ Then, add a new start symbol S.

→ Add production $S \rightarrow S_1 | S_2$.

concatenation.

→ Let L and M be CFL's with grammars G and H, respectively.

→ Assume G and H have no variables in common.

→ let S_1 and S_2 be the start symbols of G.

→ form a new grammar for LM by starting with all symbols and productions of G1 and H.

→ Add a new start symbol S.

→ Add production S → S1S2

→ Every derivation from S results in a string in L followed by one in M.

closure under star

→ Let L have grammar G1 with start symbol S1.

→ form a new grammar for L^* by introducing to G1 a new start symbol S and the productions S → S1 S | ε

→ A rightmost derivation from S generates a sequence of zero or more S1's, each of which generates some string in L.

Reversal

→ If L is a CFL with grammar G1, form a grammar for L^R by reversing the right side of every production

Ex: Let G1 have S → 0S1 | 01.

The reversal of L(G1) has grammar

S → 1S0 | 10

closed under homomorphism.

* Let L be a CFL with grammar G.
* Let h be a homomorphism on the terminal symbols of G
* construct a grammar for h(L) by replacing each termino symbol a by h(a).

'G' has production $S \to 0S1 | 01$

h is defined by $h(0) = ab$, $h(1) = \varepsilon$.

h(L(G)) has the grammar with productions

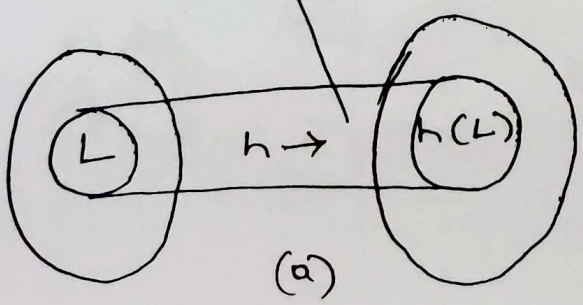
$S \to abSl | ab.$

Inverse homomorphisms:-

Suppose h is a homomorphism from some alphabet Σ to strings in another alphabet T.

Let 'L' be a language over alphabet T.

Then $h^{-1}(L)$, read "h inverse of L" is the set of strings w in Σ^* such that h(w) is in L.



(a)

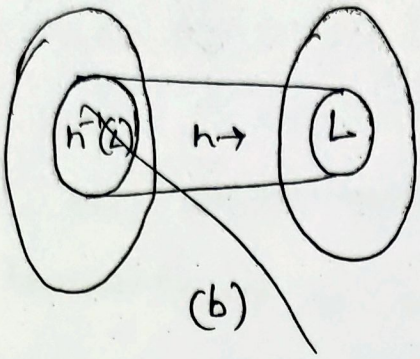


fig:- A homomorphism applied in the forward and inverse direction.

Inverse Homomorphism

$h(a) = 01$ $\quad h^{-1}(01) = a$.

$h(b) = 11$ $\quad h^{-1}(11) = b$.

$\Sigma = (0+1)^*$

$L_1 = \{\epsilon, 0, 1, 01, 11, 10, 00, 000, \ldots$
$0000, 0001, 0011, 0101, 1110, 1101 \ldots\}$ odd.

$a, b, ab, ba, aa, bb \ldots$ only.

$(a+b)^*$ $\quad (a+b)^+$

$L_1 = \{01, 11, 00, 10\}$.

$h^{-1}(L) = \{a, b\}$.

$h^{-1}(L)\{000, 001, 100, 010, 101, 111\}$.

$= \emptyset$

$L = a^n b^n$.

$h(0) = aa$

$h(1) = bb$

$h^{-1}(aa) = 0$

$h^{-1}(bb) = 1$

$L = \{a\epsilon, ab, aa, bb, aac \ldots\}$

$h^{-1}(L) = \{0, 1, 0011 \ldots\}$

↓

CFL.

$L = \{wcw^R \mid w \in (0+1)^*\}$ $\quad h(a) = 01$, $h(b) = 11$ 01

3, 7, 9, 8, 1e-2 158, 49, 48, 30129,

Chomsky Normal form

A CFG is in chomsky Normal form (CNF) if each of its productions has one of the two forms.

1. Non terminal ⟶ string of exactly two Non-termin
   i.e., $A \to BC$
2. Non terminal ⟶ one terminal, i.e., $A \to a$

Procedure for converting a Given Grammar to CNF:

1. Eliminate null & unit productions
2. Include productions of the form $A \to BC/a$ as it is.
3. Eliminate strings of terminals on the right hand side of production if they exceed one as follows:

   Suppose we have the production

   $S \to a_1 a_2 a_3$ where a_1 a_2 a_3 are terminals

   Then introduce non terminal C_{a_i} for terminal a_i is

   $C_{a_1} \to a_1$, $C_{a_2} \to a_2$, $C_{a_3} \to a_3$.

4. To restrict the number of variables on the right hand sid introduce new variables and seperate them as follows:

   suppose we have the production with n non-terminals as shown below with 5 non-terminals

   $Y \to X_1 X_2 X_3 X_4 X_5$

$4 \rightarrow X_1 R_1$

$R_1 4 \rightarrow X_2 R_2$

$R_2 4 \rightarrow X_3 R_3$

$R_3 \rightarrow X_4 R_4 X_5$

R_i where $R_1 R_2 R_3 \cdots$ are Non terminals (new).

The language generated by the new CFG is the same as that generated by the original CFG.

1. Convert the following CFG to CNF

$S \rightarrow bA \mid aB$

$A \rightarrow bAA \mid as \mid a$

$B \rightarrow aBB \mid bS \mid b$

Sol: There are no null & unit productions.

$A \rightarrow a$, $B \rightarrow b$ are in the required format.

Other productions are not in CNF.

So, replace every terminal by the following variables:

$S \rightarrow C_b A \mid C_a B$

$A \rightarrow C_b AA \mid C_a S \mid a$

$B \rightarrow C_a BB \mid C_b S \mid b$

$C_a \rightarrow a$

$C_b \rightarrow b$

$A \rightarrow C_b AA$ and $B \rightarrow C_a BB$ are the only productions that are not in CNF.

So add two new-non-terminals D and E

$C_a \rightarrow a$, $C_b \rightarrow b$

$S \rightarrow C_b A \mid C_a B$

$A \rightarrow C_b D \mid C_a S \mid a$ $\quad B \rightarrow C_a E \mid C_b S \mid b$

convert following CFG to CNF

$S \rightarrow AB \mid aB$

$A \rightarrow aab \mid \epsilon$

$B \rightarrow bbA$

Sol: Eliminate 'ε' productions.

$S \rightarrow AB \mid B \mid aB$

$A \rightarrow aab$

$B \rightarrow bbA \mid bb$

Eliminate unit production

$S \rightarrow AB \mid bbA \mid bb \mid aB$

$A \rightarrow aab$

$B \rightarrow bbA \mid bb$

Replace the terminals by a variables

$S \rightarrow AB \mid C_b C_b A \mid C_b C_b \mid C_a B$

$A \rightarrow C_a C_a C_b$

$B \rightarrow C_b C_b A \mid C_b C_b$

$D \rightarrow C_b A \quad C_a \rightarrow a$

$C_b \rightarrow b$

Introduce new non terminals D and E

$S \rightarrow AB \mid C_b D \mid C_b C_b \mid C_a B$

$A \rightarrow C_a E$

$B \rightarrow C_b D \mid C_b C_b$

$C_a \rightarrow a, \quad C_b \rightarrow b$

$D \rightarrow C_b A$

$E \rightarrow C_a C_b$

3. Convert the following CFG to CNF

$S \to ASB \mid \epsilon$, $A \to aAS \mid a$, $B \to SbS \mid A \mid bb$

Sol:-

Eliminate ϵ productions

$S \to ASB \mid AB$

$A \to aAS \mid aA \mid a$

$B \to SbS \mid Sb \mid bS \mid b \mid A \mid bb$

Eliminate unit productions

$S \to ASB \mid AB$

$A \to aAS \mid aA \mid a$

$B \to SbS \mid Sb \mid bS \mid b \mid aAS \mid aA \mid a \mid bb$

In the above Grammar

$S \to AB$.

$A \to a$

$B \to a \mid b$ are in CNF.

Replace all terminals by variable

$S \to ASB \mid AB$

$A \to C_aAS \mid C_aA \mid a$

$B \to SC_bS \mid SC_b \mid C_bS \mid b \mid C_aAS \mid C_aA \mid a \mid C_bC_b$

$C_a \to a$, $C_b \to b$.

Now

$S \to AB$

$A \to C_aA \mid a$

$B \to SC_b \mid C_bS \mid b \mid C_aA \mid C_bC_b \mid a$

$C_a \to a$, $C_b \to b$ are in CNF.

Remaining productions are

$S \rightarrow ASB$

$A \rightarrow C_a AS$

$B \rightarrow S C_b S \mid C_a AS$

Introduce new productions D and E

$D \rightarrow AS$

$E \rightarrow C_b S$

$S \rightarrow DB$

$A \rightarrow C_a D$

$B \rightarrow SE \mid C_a D$

Now All productions are in CNF the Grammar is

$S \rightarrow DB \mid AB$

$A \rightarrow C_a D \mid C_a A \mid a$

$B \rightarrow SE \mid S C_b \mid C_b S \mid b \mid C_a D \mid C_a A \mid a \mid C_b C_b$.

$C_a \rightarrow a$

$C_b \rightarrow b$

$D \rightarrow AS$, $E \rightarrow C_b S$

Greibach Normal form:

In GNF, restriction not only on the length of right sides of Productions, but also in connection with the positions at which terminals and variables can appear.

A CFL is said to be in the GNF if all the production

Converting a Given Grammar to GNF, we use two lemmas

lemma 1 (substitution rule)

let $G_1 = (V, T, P, S)$ be a CFG. Let $A \to B\alpha$ be a product in P if there is a Production $B \to B_1 | B_2 | B_3 | B_4 \ldots$.

Then the equivalent grammar can be obtained by substituting B in A. The resulting Grammar is

$$A \to B_1\alpha \mid B_2\alpha \mid B_3\alpha \mid B_4\alpha$$

lemma 2 [Elimination of left Recursion]

Grammar of the form $A \to A\alpha \mid \beta$ is called left Recursive grammar. To Eliminate left Recursion, rewrite grammar as

$$A \to \beta A'$$
$$A' \to \alpha A' \mid \epsilon$$

If we Eliminate the ϵ-production, we get

$$A \to \beta A' \mid \beta$$
$$A' \to \alpha \mid \alpha A'$$

we can generalize the Grammar, for any CFG given as

$$A \to A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \ldots \mid B_1 \mid B_2 \mid B_3 \mid \ldots$$

the Equivalent Grammar after Eliminating left recursion is

Lemmas

$$A \rightarrow B_1A' \mid B_2A' \mid B_3A' \mid B_4A' \mid \ldots \mid B_1 \mid B_2 \mid B_3 \mid B_4 \ldots$$

$$A' \rightarrow \alpha_1A' \mid \alpha_2A' \mid \alpha_3A' \mid \alpha_4A' \mid \ldots \mid \alpha_1 \mid \alpha_2 \mid \alpha_3 \ldots$$

Procedure to convert the given Grammar to GNF:

1. Eliminate null productions and unit productions, and construct CNF
2. Rename Variables as $A_1, A_2 \ldots$ starting with S as A_1
3. for each production of the form $A_i \rightarrow A_j\alpha$, apply the following:
   a) if $j > i$ .... leave the productions as they are
   b) if $j = i$ .... apply lemma2
   c) if $j < i$ .... apply lemma1.
4. for each production of the form $A_i \rightarrow A_j\alpha$, where $j > i$, apply substitution lemma if A_j is in GNF to bring A_i to GNF.

Problems

1. convert the CFG to GNF

$$S \rightarrow AA \mid a$$

$$A \rightarrow SS \mid b$$

Sol:

Rename the variables by $S \rightarrow A_1$, $A \rightarrow A_2$

$A_1 \rightarrow A_2 A_2 \mid a$ —— ①

$A_2 \rightarrow A_1 A_1 \mid b$ —— ②

for ② $j < i$ apply lemma ① ①

$A_2 \rightarrow A_1 A_1 \mid b$

$A_2 \rightarrow A_2 A_2 A_1 \mid a A_1 \mid b$

for ② apply lemma ② for this.

$A_1 \rightarrow A_2 A_2 \mid a$.

$A_2 \rightarrow \underline{A_2 A_2} A_1 \mid a A_1 \mid b$.

$A \rightarrow A\alpha \mid \beta$

$B \Rightarrow A \rightarrow$

$A_2 \rightarrow aA_1 \mid b \mid aA_1 Z \mid bZ$.

$Z \rightarrow A_2 A_1 \mid A_2 A_1 Z$

$\Rightarrow$ $A_2 \rightarrow aA_1 \mid b \mid aA_1 Z \mid bZ$

$Z \rightarrow A_2 A_1 \mid A_2 A_1 Z$

$Z \rightarrow aA_1 A_1 \mid bA_1 \mid aA_1 Z A_1 \mid bZ A_1 \mid aA_1 A_1 Z \mid bA_1 Z$

$aA_1 Z A_1 Z \mid bZ A_1 Z$

$A_1 \rightarrow aA_1 A_2 \mid bA_2 \mid aA_1 Z A_2 \mid bZ A_2$

2. Convert the following grammar G in to GNF.

$S \rightarrow XA \mid BB$

$B \rightarrow b \mid SB$

$X \rightarrow b$

$A \rightarrow a$

Sol:- Relabel the variables

$S = A_1$, $X = A_2$, $A = A_3$, $B = A_4$

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$ —— (1)

$A_4 \rightarrow b \mid A_1 A_4$ —— (2)

$A_2 \rightarrow b$ —— (3)

$A_3 \rightarrow a$ —— (4)

Apply lemma1 for (2)

$A_4 \rightarrow A_1 A_4 \mid b$.

$A_4 \rightarrow A_2 A_3 A_4 \mid A_4 A_4 A_4 \mid b$

$\because A_2 \rightarrow b$

$A_4 \rightarrow b A_3 A_4 \mid \underset{A}{A_4} \underset{A}{A_4} \underset{A}{A_4} \mid b$

lemma (2)

$A_4 \rightarrow b A_3 A_4 \mid b \mid b A_3 A_4 Z \mid bZ$

$Z \rightarrow A_4 A_4 \mid A_4 A_4 Z$

Now the resulting grammar is

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$

$A_4 \rightarrow b A_3 A_4 \mid b \mid b A_3 A_4 Z \mid bZ$

$Z \rightarrow A_4 A_4 \mid A_4 A_4 Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Still A_4 & Z are not in GNF

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

Substitute A_2 & A_4.

$A_1 \rightarrow bA_3 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

for $Z \rightarrow A_4A_4 \mid A_4A_4Z$

Substitute A_4.

$Z \rightarrow bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4 \mid bA_3A_4A_4Z \mid bA_4$

$bA_3A_4ZA_4Z \mid bZA_4Z$

final Grammarly

$A_1 \rightarrow bA_3 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

$A_4 \rightarrow bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$

$Z \rightarrow bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4 \mid bA_3A_4A_4Z \mid$

$bA_4Z \mid bA_3A_4ZA_4Z \mid bZA_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$.

3. Convert CFG to GNF.

$S \rightarrow ABA$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

Eliminate ϵ - Productions

$S \rightarrow ABA \mid BA \mid AB \mid B \mid A \mid AA$

$A \rightarrow aA \mid a$, $B \rightarrow bB \mid b$

Eliminate unit productions

$S \rightarrow ABA \mid AB \mid AA \mid aA \mid a \mid BA \mid bB \mid b$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

rename the variables

$S = A_1,\ A = A_2,\ B = A_3$

$A_1 \rightarrow A_2A_3A_2 \mid A_2A_2 \mid aA_2 \mid a \mid A_3A_2 \mid bA_3 \mid b$

$A_2 \rightarrow aA_2 \mid a$

$A_3 \rightarrow bA_3 \mid b$

Substitute A_2 & A_3 in A_1

$A_1 \rightarrow aA_2A_3A_2 \mid aA_3A_2 \mid aA_2A_2 \mid aA_2 \mid a \mid bA_3A_2 \mid bA_2 \mid bA_3 \mid b$

$A_2 \rightarrow aA_2 \mid a$

$A_3 \rightarrow bA_3 \mid b$

Decision properties of Regular languages.

membership
emptiness
equivalence

A language class is a set of languages.

* we have one example : the regular languages.
* we will see many more in this class.

language classes have two important kinds of properties

1. Decision 2. closure property

A Decision property for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.

Ex:- Is language L Empty?

description is . "the empty language" then yes, otherwise no.

but representation is a DFA (or a RE that you will convert to a DFA).

→ A closure property of a language class says that given languages in the class, an operator (e.g., union) produces another language in the same class.

Example:- the regular languages are obviously closed under union, concatenation and Kleen closure.

* up to RE representation of languages

Applications: Data validation, Data scraping, wordlist, simple parsing

convert CFG to GNF

$S \to AA|a$

$A \to SS|b$

Renaming ⇒ $A_1 \to A_2A_2|a$ — ①

$A_2 \to A_1A_1|b$ — ②

$i > j$ - substitution rule.

$A_2 \to A_2A_2A_1|aA_1|b$

α β_1 β_2

use lemma-2,

$A \to A\alpha|\beta$ ⇒ $A \to \beta A'|\beta$.
$A' \to \alpha A'|\alpha$.

$A_2 \to aA_1Z|aA_1|b|bZ$

$Z \to A_2A_1Z|A_2A_1$

$Z \to aA_1ZA_1Z|aA_1A_1Z|bA_1Z|bZA_1Z$
$aA_1ZA_1|aA_1A_1|bA_1|bZA_1$

$A_1 \to A_2A_2|a$.

$j > i$:-

$A_1 \to aA_1ZA_2|aA_1A_2|bA_2|bZA_2|a$.

✓

---

4 5|3|12

[illegible], F9, [illegible], F8, [illegible], G7, H5, [illegible],

ie- 25, [illegible], 34,

6|3|12 → [illegible], H5, H9, [illegible] ie- 25[illegible], 28, 34,

$S \to \cdot XA \mid BB$

$B \to b \mid SB$

$X \to b$

$A \to a$

$S, X, A, B \Rightarrow A_1, A_2, A_3, A_4$

$A_1 \to A_2A_3 \mid A_4A_4$

$A_4 \to b \mid A_1A_4$ ✓

$A_2 \to b$

$A_3 \to a$

$A_4 \to A_1A_4 \mid b$

$i > j$ — terminal — ①

$A_4 \to A_2A_3A_4 \mid \cancel{A_4A_4A_4} \mid A_4A_4A_4 \mid b$

Consider $A_4 \to A_2A_3A_4$

$A_4 \to bA_3A_4$

$A_4 \to A_4A_4A_4 \mid b$

$A \quad \alpha \quad \beta_1$

$A_4 \to \cancel{B_1}\, bZ \mid b \mid bA_3A_4Z$

$Z \to A_4A_4Z \mid A_4A_4$

$Z \to bZA_4Z \mid bA_4Z \mid bZA_4 \mid bA_4$

$\to bA_3A_4A_4Z \mid bA_3A_4A_4$

$A_1 \rightarrow A_2 A_3$

$A_1 \rightarrow bA_3.$

$A_1 \rightarrow A_4 A_4.$

$A_4 \rightarrow bA_3 A_4 \mid bz \mid b.$

$j > i.$

$A_1 \rightarrow bA_3 A_4 A_4 \mid bzA_4 \mid bA_4 \mid bA_3.$

$A_4 \rightarrow bA_3 A_4 \mid bz \mid b.$

$z \rightarrow bzA_4 z \mid bA_4 z \mid bzA_4 \mid bA_4 \mid bA_3 A_4 A_4 z$

$bA_3 A_4 A_4.$

$A_2 \rightarrow b.$

$A_3 \rightarrow a.$

3/3/18

[illegible], C5, C9, D2, D3, D6, D7, D9, [illegible], E2, E4, E6, E8, F0, F1

F6, F9, G0, G1, G3, [illegible], H2, H3, H4, H5, [illegible], H8

H9, J0, T-86, Le - [illegible], 28, [illegible], 32, 34,

GNF

In GNF, we put restrictions not only on the length of right sides of production, but also in connection with the positions at which terminals and variables can appear.

Def:- A CFG is said to be in GNF if all the productions are of the form $A \to a\alpha$ where $a \in T$, $\alpha \in V^*$.

2 lemmas (1- substit—)

$A \to B\alpha$

$B \to \beta_1 | \beta_2 | \beta_3 \ldots | \beta_n$

then $A \to \beta_1\alpha | \beta_2\alpha | \beta_3\alpha \ldots$

lemma-2

$A \to A\alpha | \beta$

$\Rightarrow$ $A \to \beta A'$, $A' \to \alpha A' | \epsilon$ } $A \to \beta A' | \beta$, $A' \to \alpha A' | \alpha$

$A \to A\alpha_1 | A\alpha_2 | A\alpha_3 \ldots | \beta_1 | \beta_2 | \beta_3$

$A \to \beta_1 A' | \beta_2 A' | \beta_3 A' | \ldots | \beta_1 | \beta_2 | \beta_3 \ldots$

$A' \to \alpha_1 A' | \alpha_2 A' | \alpha_3 A' | \ldots | \alpha_1 | \alpha_2 | \alpha_3 \ldots$

1 → simplify

2 → Rename

3 → chek the condition for 'j & i'

If $j > i$ → leave

$j = i$ → lemma (2) ; $i > j$ → lemma (1).

4 → if $j > i$ if in GNF —