NPR College of Engineering & Technology
NPR Nagar, Natham, Dindigul - 624401, Tamil Nadu, India.
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai.
An ISO 9001:2015 Certified Institution.
Phone No: 04544- 246 500, 246501, 246502.
Website : www.nprcolleges.org, www.nprcet.org, Email:nprcetprincipal@nprcolleges.org

CS8501

THEORY OF COMPUTATION

by,
C.Kalpana, M.E(CSE)
Assistant Professor,
Department of Computer Science and Engineering,
NPR College of Engineering & Technology

Syllabus

UNIT IV PROPERTIES OF CONTEXT FREE LANGUAGES

Normal Forms for CFG – Pumping Lemma for CFL – Closure Properties of CFL – Turing Machines – Programming Techniques for TM.

Definition

CONTEXT FREE LANGUAGE (CFL)

* The language generated by the Context free Grammar is called Context free language.

PROPERTIES OF CFL

i) Simplify the Context Free Grammar

ii) Pumping Lemma (Proove the language is not to be CFL)

iii) Closure & Decission Properties => Proove and produce

Simplification of Grammar

CFG has Single Non terminal on the left hand side and any no. of terminals and Non-terminals on the Right Hand side.

Ex:

LHS	RHS
A	aBB
B	abC
C	d / ε

Where,

ε - Null production

It Includes

i) Elimination of Useless Symbol
ii) Elimination of ε- production
iii) Eliminating Unit production.

These are the Normal forms of Context Free Grammar (CFG).

Elimination of Useless Symbol

DEFINITION

A Symbol is useless, if it cannot derive a terminal or it is not reachable from the Start Symbol.

1) if $X \overset{*}{\Rightarrow} \omega$ for some terminal string, ω

[Derives terminal string]

2) X is reachable if there is a derivation

$S \overset{*}{\Rightarrow} \alpha X \beta$ for some α and β.

Where α & β are terminals.

Eliminate ⇒ Not Generate a terminal & Not used Symbol.

Examples

1) Consider the CFG $G=(V,T,P,S)$

Where, $V=\{S,A,B\}$

$T=\{0,1\}$ $P=\{S\rightarrow A11B \mid 11A$

$S\rightarrow B \mid 11$, $A\rightarrow 0$, $B\rightarrow BB\}$

Remove Useless Symbol.

Solution

Step1 Eliminate the symbols which are not generating terminals.

$S\rightarrow 11A$ $[\because A\rightarrow 0]$

$A\rightarrow 0$

Step2 Remove the unused Symbol.

Since B has no productions

So We need to Eliminate B.

Ans

$S\rightarrow 11A \mid 11$

$A\rightarrow 0$

Remove the Useless Symbols from the following grammar.

$S \rightarrow aA / bB$

$A \rightarrow aA / a$

$B \rightarrow bB$

$D \rightarrow ab / Ea$

$E \rightarrow aC / d$

Solution

Step 1 Eliminate Unused term

D and E are Unused Symbols.

$S \rightarrow aA / bB$

$A \rightarrow aA / a$

$B \rightarrow bB.$

Step 2 Eliminate the Non-generating terminals

$S \rightarrow aA$

$A \rightarrow aA / a$

[∵ B has no terminals So we Eliminate B term.]

Elimination of Epsilon Productions

If there is ε-production we can remove it, without changing the meaning of the grammar.

Nullable Variable

In a given CFG, a Nonterminal X is nullable, if,

i) there is a production $X \to \varepsilon$

ii) There is a derivation that starts at X and leads to ε.

$X \Rightarrow * \ldots \Rightarrow \varepsilon$

i.e., $X \overset{*}{\Rightarrow} \varepsilon$

Examples

Ex: 1

Eliminate the null production in the grammar.

$S \rightarrow ABaC$

$A \rightarrow BC$

$B \rightarrow b / \varepsilon$

$C \rightarrow D / \varepsilon$

$D \rightarrow d$.

Solution

Eliminate ε-productions

$B \rightarrow b$

$C \rightarrow D$

$D \rightarrow d$

Since

$S \rightarrow ABaC \rightarrow B\varepsilon aC \rightarrow ba$

$A \rightarrow BC \rightarrow bC \rightarrow bD \rightarrow bd$

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

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

2) Eliminate ε-productions

$S \rightarrow 0S / 1S / \varepsilon$

Sol

Step 1 Eliminate ε production

$S \rightarrow 0S$ [since $S \rightarrow \varepsilon$]

$S \rightarrow 0$

$S \rightarrow 1S$ [since $S \rightarrow \varepsilon$]

$S \rightarrow 1$

Solution

$S \rightarrow 0S / 1S / 0 / 1$

Elimination of Unit Productions

A Unit production is a production of the form $A \rightarrow B$.

Where,

Both A and B are Variables.

Unit Pair

* If the sequence of derivation steps are,

$A \Rightarrow B_1 \Rightarrow B_2 \Rightarrow B_3 \cdots B_n \Rightarrow \alpha$

Then these Unit Productions are replaced by a non-Unit production

i.e.,

$B_n \Rightarrow \alpha$ directly from A

i.e., $A \rightarrow \alpha$

$\therefore (A, B)$ such that $A \overset{*}{\Rightarrow} B$ is called an Unit pair.

Examples

) Eliminate Unit Production

$S \rightarrow OAB$, $A \rightarrow 01 / B$, $B \rightarrow 0A / 1$

SOLUTION

Step1 Here, only one Unit production i.e, B.

Step2

$S \rightarrow OAB$, $B \rightarrow 0A / 1$, $A \rightarrow 01$

Replace the production by finding Unit pairs.

$A \rightarrow B \rightarrow 0A$ [$\because B \rightarrow 0A / 1$]

$A \rightarrow B \rightarrow 1$

Step3 The equivalent production without Unit

Production is given by,

$S \rightarrow OAB$

$A \rightarrow 01 / 0A / 1$

$B \rightarrow 0A / 1$

Eliminate Unit production

$S \to 0A \mid 1B \mid C$, $B \to 1 \mid A$

$A \to 0S \mid 00$, $C \to 01$

Solution

Step1 Here only one Unit production i.e, C

$S \to 0A \mid 1B$, $A \to 0S \mid 00$, $C \to 01$

$\boxed{S \to C}$ → Unit production

Step2 Replace the Unit production

$B \to A \to 0S \mid 00$ [$\because A \to 0S \mid 00$]

$S \to C \to 01$ [$\because C \to 01$]

i) $S \to 01$ //

Step3 The equivalent grammar without Unit production is,

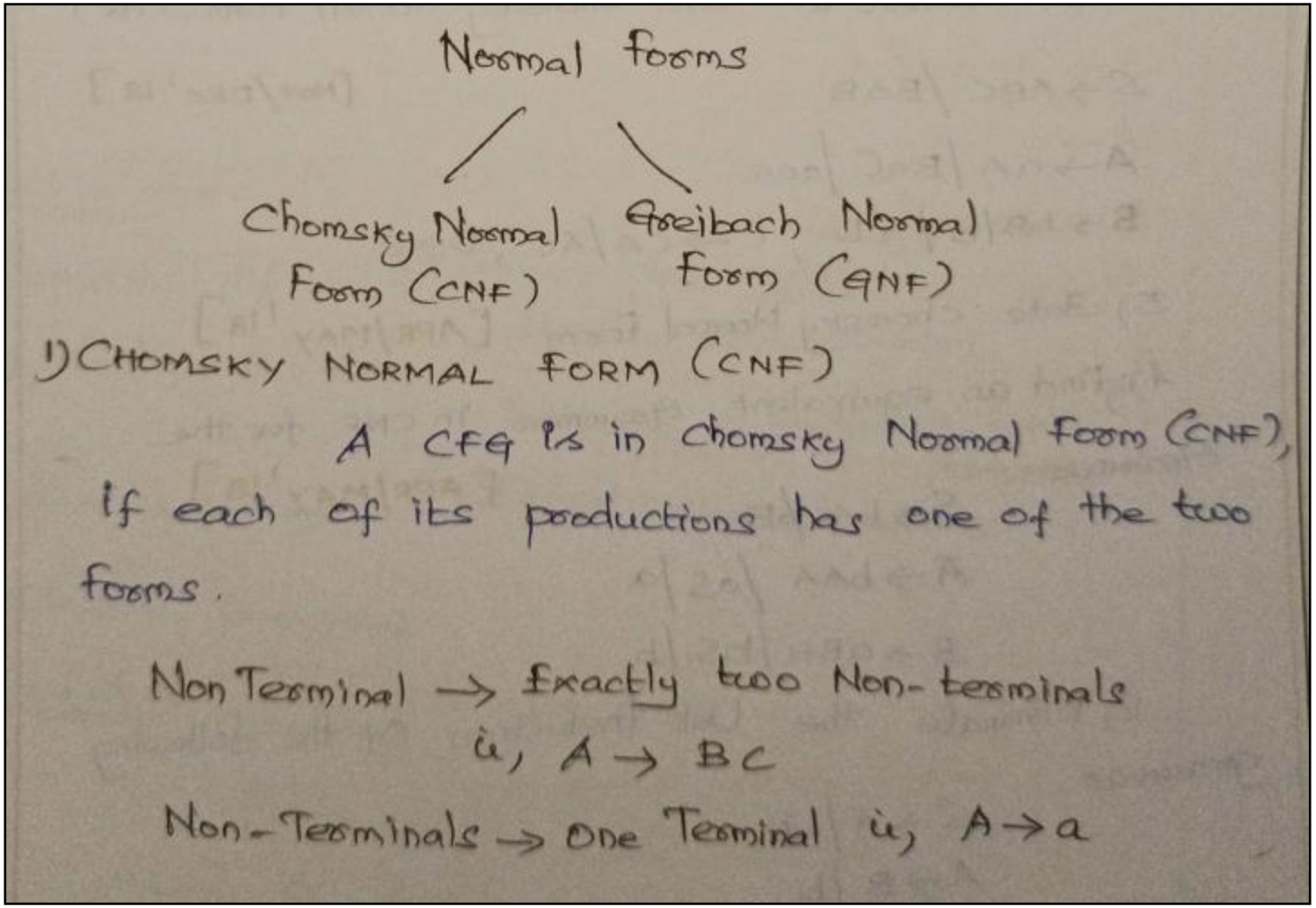
Ans $S \to 0A \mid 1B \mid 01$

$A \to 0S \mid 00$

$B \to 1 \mid 0S \mid 00$

$C \to 01$

Normal Forms of CFG



CNF Format (Rules)

$NT \rightarrow NT \cdot NT$

$NT \rightarrow T$

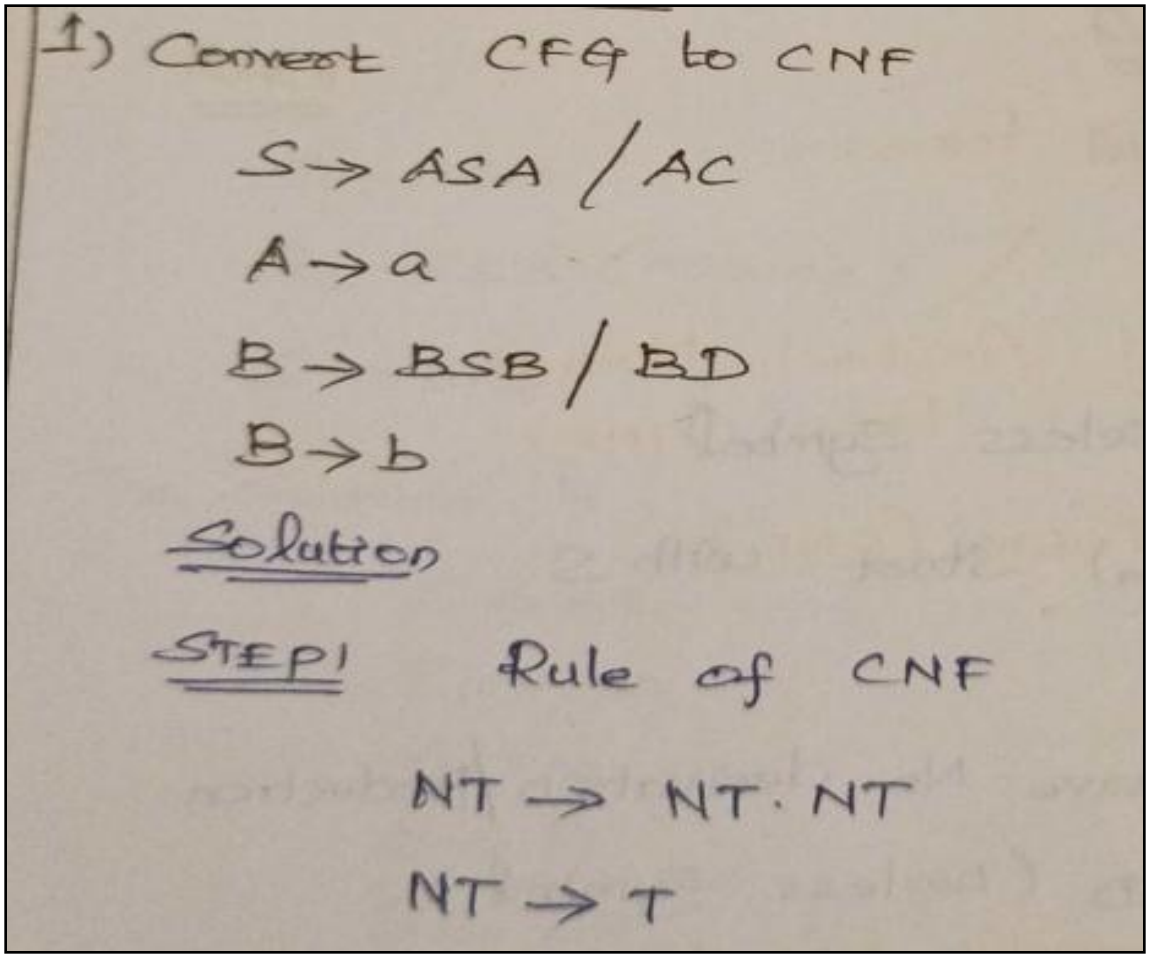
Where, NT - Non-Terminal

T - Terminal

STEPS TO SOLVE CNF PROBLEM

1) Rule
2) Given CFG
3) Simplified CFG
4) Convert simplified CFG to CNF

Example Problems



STEP2 Given CFG

$S \rightarrow ASA / AC$, $B \rightarrow BSB / BD$

$A \rightarrow a$, $B \rightarrow b$

STEP3 Simplified CFG →

i) Elimination of ε - Production

There is no ε - production in the given grammar.

$S \rightarrow ASA / AC$

$A \rightarrow a$

$S \rightarrow BSB / BD$

$B \rightarrow b$

ii) Elimination of Unit Production

There is no Unit Production

$S \rightarrow ASA \mid AC$

$A \rightarrow a$

$S \rightarrow BSB \mid BD$

$B \rightarrow b$

iii) Elimination of Useless Symbol

All Non-Terminal start with S

$NT \rightarrow T$

Since C & D have No derivation/production

Eliminate C & D (useless symbol)

$S \rightarrow ASA$

$A \rightarrow a$

$B \rightarrow BSB$

$B \rightarrow b$

Step 4

Convert simplified CFG to CNF

Given CFG,

$S \rightarrow ASA$ (Apply Rule1)

Because here S has More than one Non terminal

$A \rightarrow a$ //Already CNF Rule2

$S \rightarrow BSB$ (Apply Rule3)

$B \rightarrow b$ //Already CNF (Rule4)

Rule1

$S \rightarrow ASA$

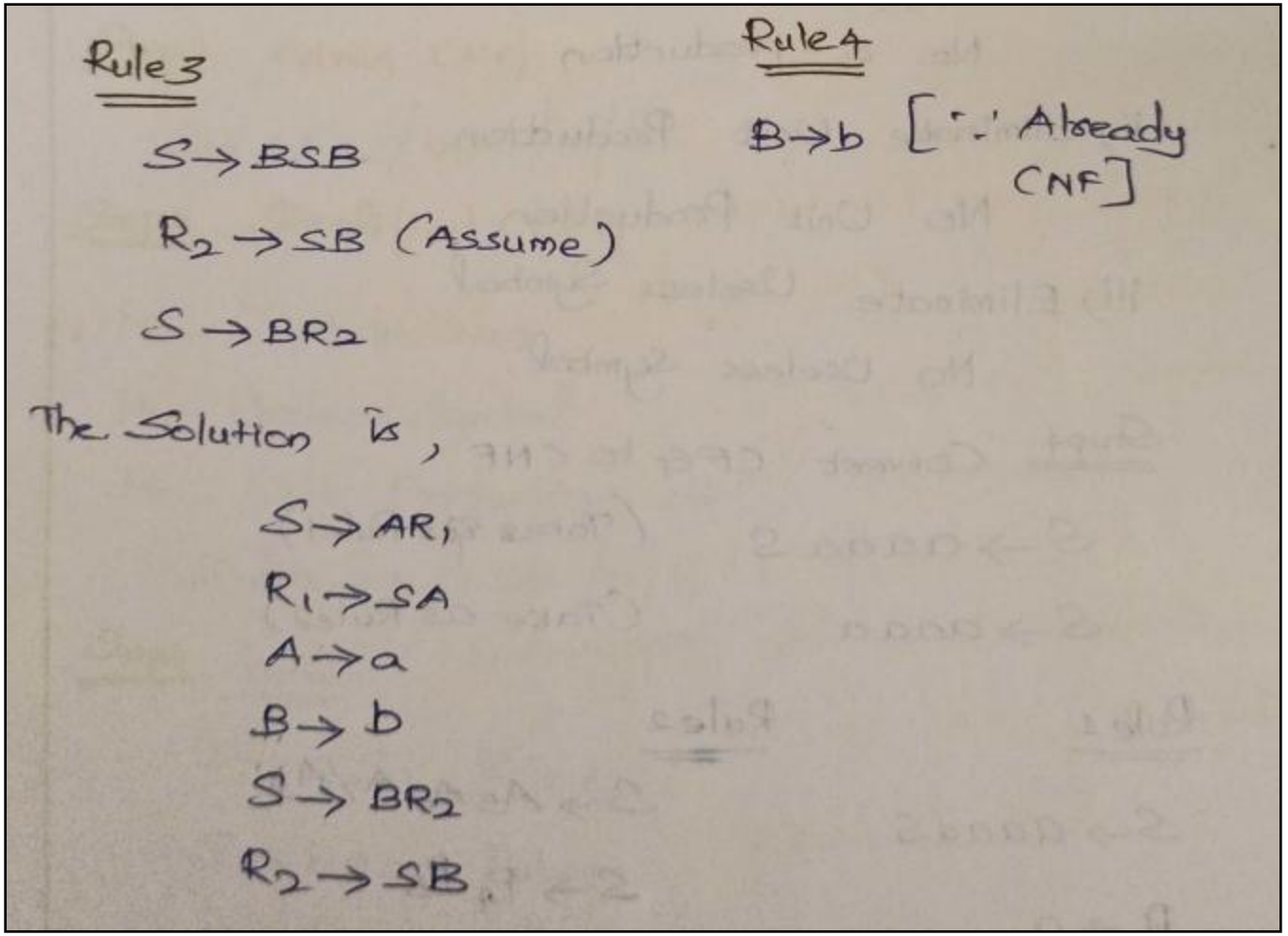
$R_1 \rightarrow SA$ (Assume)

Then,

$S \rightarrow AR_1$

Rule 2

$A \rightarrow a$ [∵ Already CNF]


Rule 3
S → BSB
R2 → SB (Assume)
S → BR2
The Solution is,
S → AR1
R1 → SA
A → a
B → b
S → BR2
R2 → SB
Rule 4
B → b [∵ Already CNF]

2) Convert CFG to CNF

$S \rightarrow aaaaS$

$S \rightarrow aaaa$

Solution

Step1 Rule CNF

$NT \rightarrow NT \cdot NT$

$NT \rightarrow T$

Step2 Given CFG

$S \rightarrow aaaaS$

$S \rightarrow aaaa$

Step3

i) Eliminate ε - production

No ε - production

ii) Eliminate Unit Production

No Unit Production

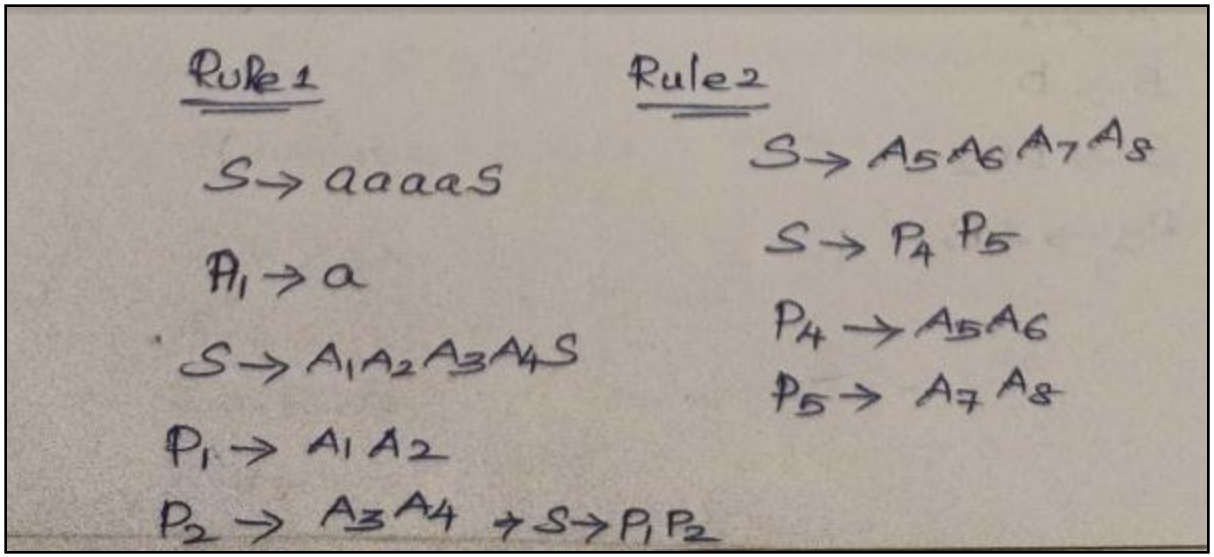
iii) Eliminate Useless Symbol

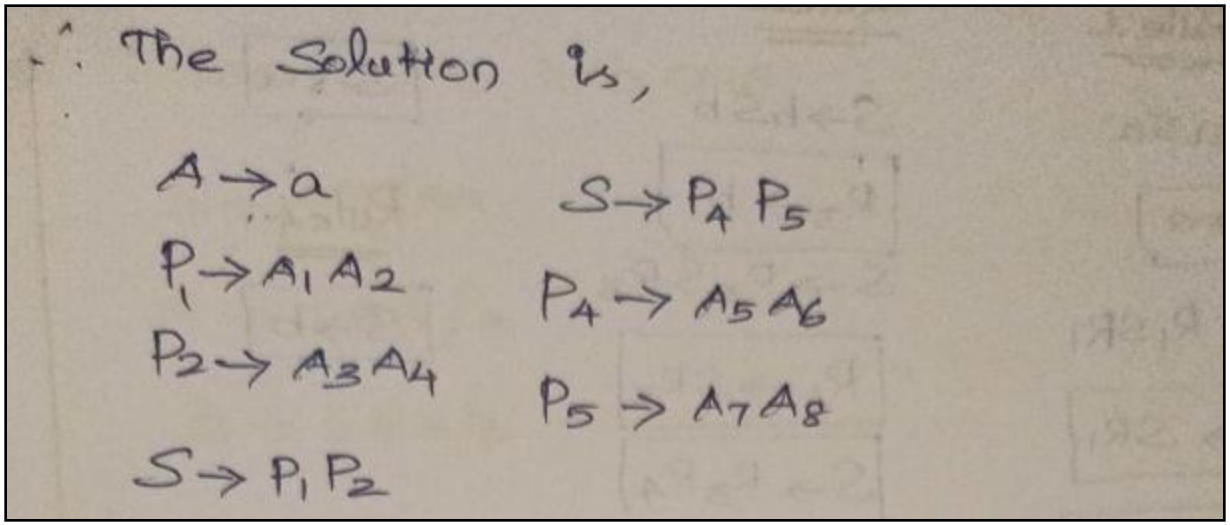
No Useless Symbol.

Step1 Convert CFG to CNF

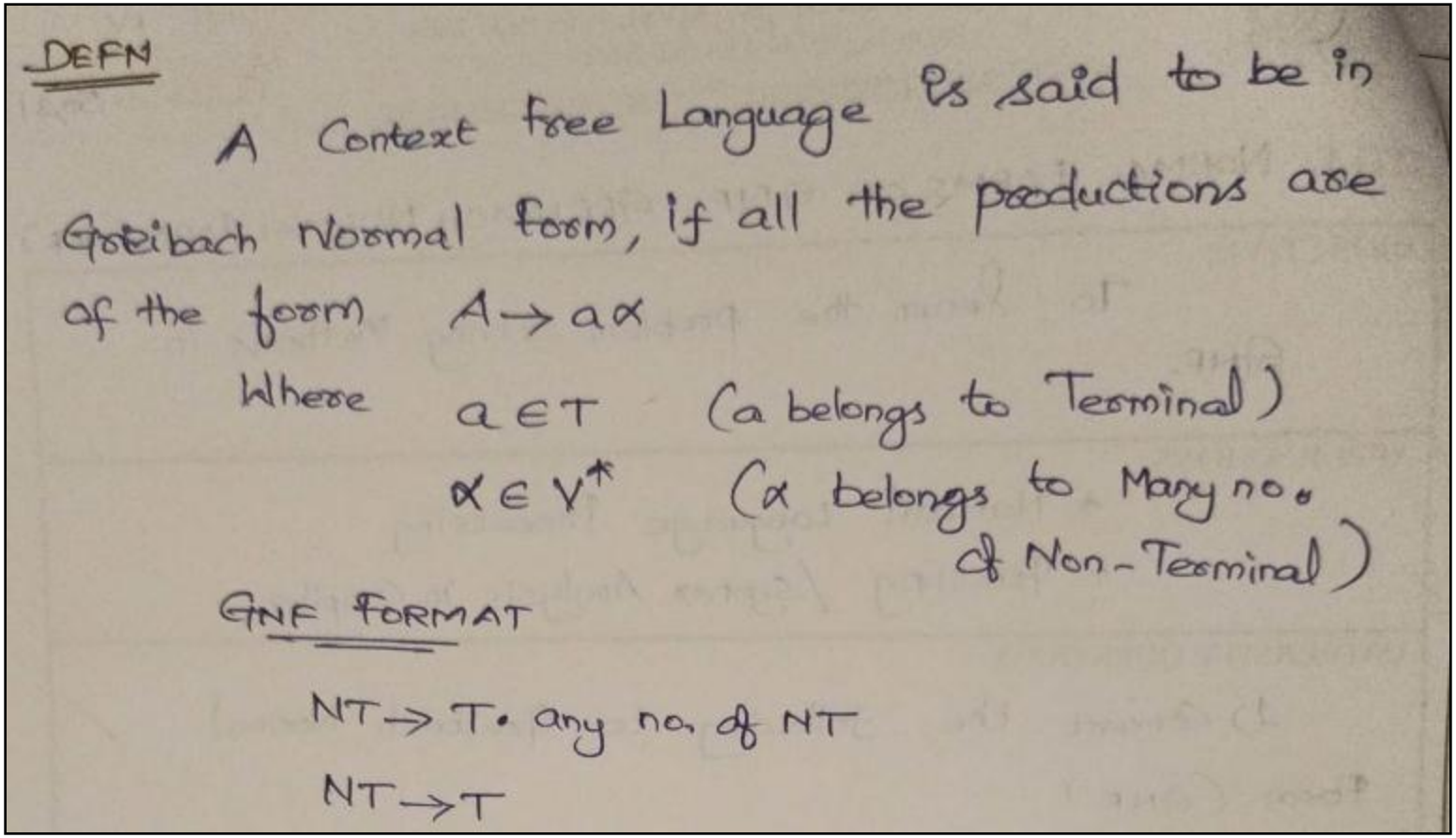
$S \rightarrow aaaaS$ (Takes as Rule1)

$S \rightarrow aaaa$ (Take as Rule2)


Rule 1
S → aaaaS
A1 → a
S → A1A2A3A4S
P1 → A1A2
P2 → A3A4 → S → P1P2
Rule 2
S → A5A6A7A8
S → P4P5
P4 → A5A6
P5 → A7A8


∴ The Solution is,
A → a
P1 → A1A2
P2 → A3A4
S → P1P2
S → P4P5
P4 → A5A6
P5 → A7A8

Greibach Normal Forms



For example :

S → aA
S → a
} is in GNF

But S → AA
Or S → Aa
} is not in GNF

To convert given CFG into GNF very interesting procedure is followed.

Step 1 : Convert the given grammar into simplified form by eliminationg ε productions, unit productions and useless symbols.

Step 2 : Bring the grammar to Chomsky's Normal Form (CNF). This step can be optional.

Step 3 : Number the non-terminal symbols in ascending order i.e. A_1, A_2, A_3 and so on.

Step 4 : In the rule of the form

$$A_i \rightarrow A_j A_k \ldots$$

There must be $i < j$. If $i > j$ or $i = j$ then process the rule to convert it to GNF.

When $i = j$ i.e.

$$A_i \rightarrow A_j A_k \ldots$$

then that grammar is said to be **left recursive**. Remove the left recursion using following formula

If $A \rightarrow A\alpha | \beta$

then convert above grammar in the following form

$$A \rightarrow \beta A' | \beta$$

$$A' \rightarrow \alpha A' | \alpha$$

Example Problems

1) Convert CFG to GNF

$A_1 \rightarrow A_2 A_3$

$A_2 \rightarrow A_3 A_1 \mid b$

$A_3 \rightarrow A_1 A_2 \mid a$

Solution

STEP1 Rule of GNF

$NT \rightarrow T.$ any no. of NT

$NT \rightarrow T$

STEP2 Given CFG

$A_1 \rightarrow A_2 A_3$

$A_2 \rightarrow A_3 A_1 \mid b$

$A_3 \rightarrow A_1 A_2 \mid a$

STEP 3 : Simplification of CFG

There is no ε (Epsilon) production, useless production and Unit production.

The Grammar is,

$A_1 \rightarrow A_2 A_3$

$A_2 \rightarrow A_3 A_1 / b$

$A_3 \rightarrow A_1 A_2 / a$

STEP4 Simplify CFG to GNF

i) If $i<j$

$A_1 \rightarrow A_2 A_3$

$A_2 \rightarrow A_3 A_1 / b$

ii) if $i>j$ (lemma 1)

$A_3 \rightarrow A_1 A_2 / a$

Sub. A_1 in A_3

$A_3 \rightarrow A_2 A_3 A_2 / a$

Again Check the Condition

Sub. A_2 in A_3

$A_3 \rightarrow A_3 A_1 A_3 A_2 / b A_3 A_2 / a$

Again check $i>j$ (false)

Check $i=j$ (true)

Apply Lemma

$B \rightarrow A_1 A_3 A_2$

$B \rightarrow A_1 A_3 A_2 B$

Ex:

$i<j$ means,

$A_1 \rightarrow A_2 A_3$

i) $1<2$

$A_2 \rightarrow A_3 A_1$

ii) $2<3$

Ex: $i>j$

$A_3 \rightarrow A_1 A_2$

i) $3>1$.

$A_3 \rightarrow bA_3A_2 / a$
$A_3 \rightarrow bA_3A_2B / aB$ } GNF

Sub. A_3 in A_2

$A_2 \rightarrow bA_3A_2A_1 / aA_1 / b / bA_3A_2BA_1 / aBA_1$

Sub A_2 in A_1

$A_1 \rightarrow bA_3A_2A_1A_3 / aA_1A_3 / bA_3 / bA_3A_2BA_1A_3 /$
$aBA_1A_3.$

Sub. A_1 in B

$B \rightarrow bA_3A_2A_1A_3A_3A_2 / aA_1A_3A_3A_2 / bA_3A_3A_2 /$
$bA_3A_2BA_1A_3A_3A_2 / aBA_1A_3A_3A_2.$

$B \rightarrow bA_3A_2A_1A_3A_3A_2B / aA_1A_3A_3A_2B /$
$bA_3A_3A_2B / bA_3A_2BBA_1A_3A_2B /$
$aBA_1A_3A_3A_2B$

∴ The solutions are,
(GNF)

$A_1 \rightarrow bA_3A_2A_1A_3 / aA_1A_3 / bA_3 / bA_3A_2BA_1A_3 / aBA_1A_3$

$A_2 \rightarrow bA_3A_2A_1 / aA_1 / b / bA_3A_2BA_1 / aBA_1$

$A_3 \rightarrow bA_3A_2 / a / bA_3A_2B / aB$.

$B \rightarrow bA_3A_2A_1A_3A_3A_2 / aA_1A_3A_3A_2 / bA_3A_3A_2 / bA_3A_2BA_1A_3A_3A_2 / aBA_1A_3A_3A_2$

$B \rightarrow bA_3A_2A_1A_3A_3A_2B / aA_1A_3A_3A_2B / bA_3A_3A_2B / bA_3A_2BA_1A_3A_3A_2B / aBA_1A_3A_3A_2B$.

Closure properties of CFL

- **Closure properties** consider operations on CFL that are guaranteed to produce a CFL
- The CFL's are closed under *substitution*, *union*, *concatenation*, *closure (star)*, *reversal*, *homomorphism* and *inverse homomorphism*.
- CFL's are not closed under *intersection* (but the intersection of a CFL and a regular language is always a CFL), *complementation*, and *set-difference*.

Substitution

- Each symbol in the strings of one language is replaced by an entire CFL language
- Useful in proving some other closure properties of CFL
- Example: $S(0) = \{a^n b^n | \ n \geq 1\}$, $S(1) = \{aa, bb\}$ is a substitution on alphabet $\Sigma = \{0, 1\}$.

Substitution

- **Theorem:** If a substitution s assigns a CFL to every symbol in the alphabet of a CFL L, then s(L) is a CFL.
- **Proof**
  - Let G = (V, $\sum$, P, S) be grammar for L
  - Let G_a= (V_a, T_a, P_a, S_a) be the grammar for each $a \in \sum$ with $V \cap V_a = \phi$
  - G′= (V′, T′, P′, S) for s(L) where
    - V′ = $V \cup V_a$
    - T′ = union of T_a for all $a \in \sum$
    - P′ consists of
      - » All productions in any P_a for $a \in \sum$
      - » In productions of P, each terminal a is replaced by S_a
- A detailed proof that this construction works is in the reader.
  - Intuition: this replacement allows anystring in La to take the place of any occurrence of a in any string of L.

Example (1)

- L = $\{0^n1^n | \ n \geq 1\}$, generated by the grammar $S \rightarrow 0S1|01$,
- s(0) = $\{a^nb^m | m \leq n\}$, generated by the grammar $S \rightarrow aSb|A$; $A \rightarrow aA|\ ab$,
- s(1)={ab, abc}, generated by the grammar $S \rightarrow abA$, $A \rightarrow c|\varepsilon$
- Rename second and third S's to S_0 and S_1 respectively. Rename second A to B. Resulting grammars are:

  $S \rightarrow 0S1 \mid 01$

  $S_0 \rightarrow aS_0b \mid A$; $A \rightarrow aA \mid ab$

  $S_1 \rightarrow abB$; $B \rightarrow c \mid \varepsilon$

Example(1) Continution

- In the first grammar replace 0 by S_0 and 1 by S_1. The combined grammar:

  $G' = (\{S, S_0, S_1, A, B\}, \{a, b\}, P', S)$,

  where $P' = \{S \rightarrow S_0SS_1 \mid S_0S_1, S_0 \rightarrow aS_0b \mid A, A \rightarrow aA \mid ab, S_1 \rightarrow abB, B \rightarrow c \mid \varepsilon\}$

Application of Substitution

- Closure under union of CFL's L_1 and L_2
- Closure under concatenation of CFL's L_1 and L_2
- Closure under Kleene's star (closure * and positive closure $^+$) of CFL's L_1
- Closure under homomorphism of CFL L_i for every $a_i \in \sum$

Union

- Use L= {a, b}, s(a) = L_1 and s(b)=L_2.s(L)= $L_1 \cup L_2$
- To get grammar for $L_1 \cup L_2$?
  - Add new start symbol S and rules $S \rightarrow S_1|S_2$
  - We get grammar G = (V, T, P, S) where

    $V = V_1 \cup V_2 \cup \{ S \}$, where $S \notin V_1 \cup V_2$

    $P = P_1 \cup P_2 \cup \{ S \rightarrow S_1 \mid S_2 \}$
- Example:
  - $L_1 = \{ a^n b^n \mid n \geq 0 \}$, $L_2 = \{ b^n a^n \mid n \geq 0 \}$
  - $G_1 : S_1 \rightarrow aS_1b \mid \varepsilon$, $G_2 : S_2 \rightarrow bS_2a \mid \varepsilon$
  - $L1 \cup L2$ is G = ($\{S_1, S_2, S\}$, {a, b}, P, S) where P = {$P1 \cup P2 \cup \{S \rightarrow S_1 \mid S_2 \}$}

Concatenation

- Let L={ab}, s(a)=L_1 and s(b)=L_2. Then s(L)=L_1L_2
- To get grammar for L_1L_2 ?
  - Add new start symbol and rule $S \rightarrow S_1S_2$
  - We get G = (V, T, P, S) where

    $V = V_1 \cup V_2 \cup \{ S \}$, where $S \notin V_1 \cup V_2$

    $P = P_1 \cup P_2 \cup \{ S \rightarrow S_1S_2 \}$
- Example:
  - $L_1 = \{ a^nb^n \mid n \geq 0 \}$ with G_1: $S_1 \rightarrow aS_1b \mid \varepsilon$
  - $L_2 = \{ b^na^n \mid n \geq 0 \}$ with $G_2 : S_2 \rightarrow bS_2a \mid \varepsilon$
  - $L_1L_2 = \{ a^nb^{\{n+m\}}a^m \mid n, m \geq 0 \}$ with $G = (\{S, S_1, S_2\}, \{a, b\}, \{S \rightarrow S_1S_2, S_1 \rightarrow aS_1b \mid \varepsilon, S_2 \rightarrow bS_2a\}, S)$

Kleene's star

- Use L={a}* or L={a}$^+$, s(a)=L_1. Then s(L)=L_1* (or s(L)=L_1^+).
- Example:
  - $L_1 = \{a^n b^n \mid n \geq 0\}$ $(L_1)^* = \{ a^{\{n1\}}b^{\{n1\}} \ldots a^{\{nk\}}b^{\{nk\}} \mid k \geq 0$ and $ni \geq 0$ for all i $\}$
  - $L_2 = \{ a^{\{n^2\}} \mid n \geq 1 \}$, $(L_2)^* = a^*$
- To get grammar for $(L_1)^*$
  - Add new start symbol S and rules $S \rightarrow SS_1 \mid \varepsilon$.
  - We get G = (V, T, P, S) where

    $V = V_1 \cup \{ S \}$, where $S \notin V_1$

    $P = P_1 \cup \{ S \rightarrow SS_1 \mid \varepsilon \}$

Homomorphism

- Closure under homomorphism of CFL L for every $a \in \sum$
- Suppose L is a CFL over alphabet $\sum$ and h is a homomorphism on $\sum$.
- Let s be a substitution that replaces every $a \in \sum$, by h(a). ie s(a) = {h(a)}.
- Then h(L) = s(L).
- $h(L) = \{h(a_1)\ldots h(a_k) \mid k \geq 0\}$ where $h(a_k)$ is a homomorphism for every $a_k \in \sum$.

Reversal

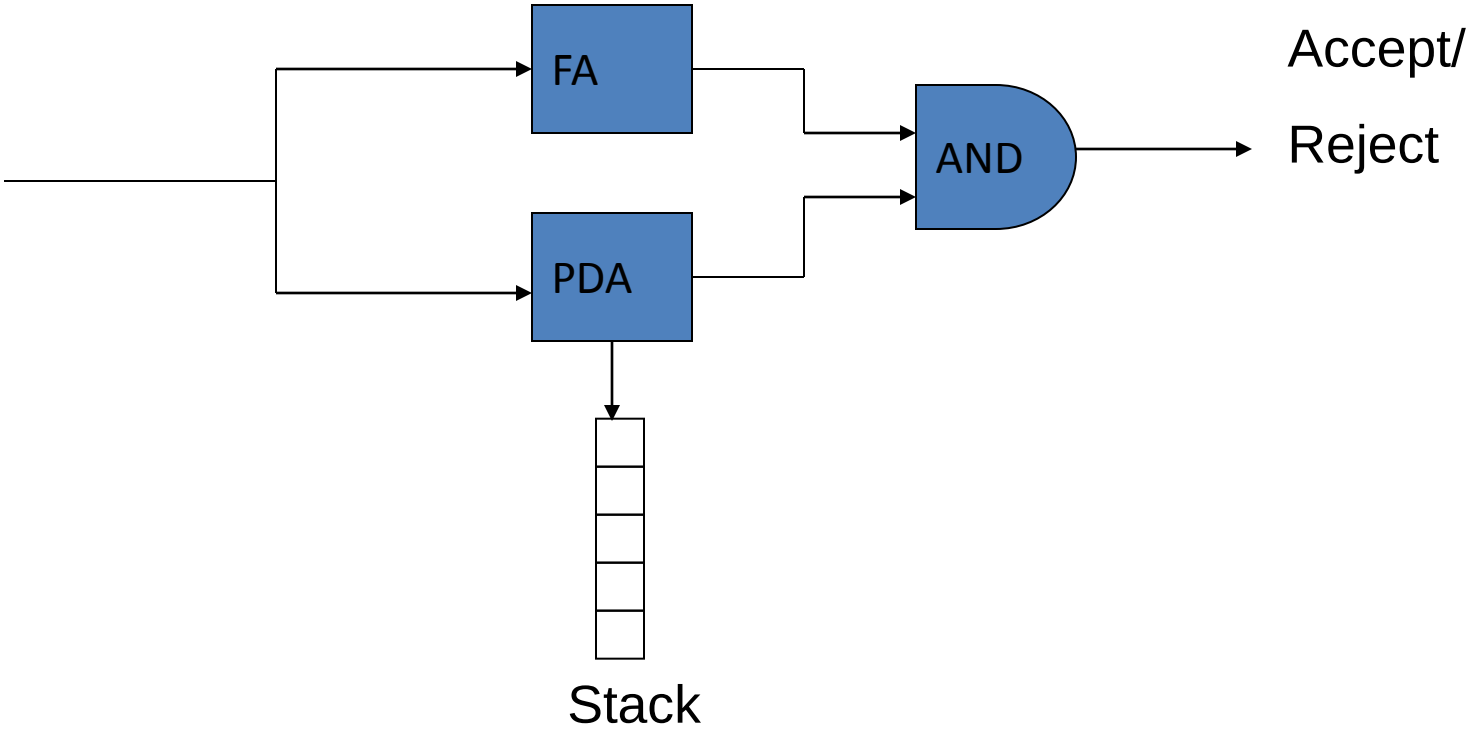
- The CFL's are closed under reversal
- This means then if L is a CFL, so L^R is a CFL
- It is enough to reverse each production of a CFL for L, i.e., substitute $A \rightarrow \alpha$ by $A \rightarrow \alpha^R$
- Example:
  - $L = \{ a^n b^n \mid n \geq 0 \}$ with $P : S \rightarrow aSb \mid \varepsilon$
  - $L^R = \{ b^n a^n \mid n \geq 0 \}$ with $P^R : S \rightarrow bSa \mid \varepsilon$

Intersection

- The CFL's are not closed under intersection
- Example:
  - L = $\{0^n1^n2^n | n \geq 1\}$ is not context-free.
  - L1 = $\{0^n1^n2^i | n \geq 1, i \geq 1\}$, L2 = $\{0^i1^n2^n | n \geq 1, i \geq 1\}$ are CFL's with corresponding grammars for L1: S->AB; A->0A1 | 01; B->2B | 2 , and for L2: S ->AB; A->0A | 0; B->1B2 | 12.
  - However, $L = L_1 \cap L_2$
  - Thus intersection of CFL's is not CFL

Intersection with RL

- Theorem: If L is CFL and R is a regular language, then L ∩ R is a CFL.



Intersection with RL

- $P=(Q_P, \sum, \Gamma, \delta_P, q_P, Z_0, F_P)$ be PDA to accept CFL by final state
- $A=(Q_A, \sum, \delta_A, q_A, F_A)$ be a DFA for RL
- Construct PDA $P' = (Q, \sum, \Gamma, \delta, q_o, Z_0, F)$ where
  - $Q = Q_p \text{ X } Q_A$
  - $q_o = (q_p, q_A)$
  - $F = (F_P \text{ X } F_A)$
  - δ is in the form $\delta((q, p), a, X) = ((r, s), \gamma)$ such that
    1. $s = \delta_A(p, a)$
    2. (r, γ) is in $\delta_P(q, a, X)$

- For each move of PDA P, we make the same move in PDA P′ and also we carry along the state of DFA A in a second component of P′.
- P′ accepts a string w iff both P and A accept w.
- w is in $L \cap R$.
- The moves $((q_p, q_A), w, Z) \vdash^* P' ((q, p), \varepsilon, \gamma)$ are possible iff $(q_p, w, Z) \vdash^* P (q, \varepsilon, \gamma)$ moves and $p = \delta^*(q_A, w)$ transitions are possible.

Set Difference with RL

- For a CFL's L, and a regular language R.

  L - R is a CFL.

  Proof:
  - R is regular and R^C is also regular
  - $L - R = L \cap R^C$
  - Complement of of Regular Language is regular
  - Intersection of a CFL and a regular language is CFL

Complementation

- L^C is not necessarily a CFL
- Proof:
  - Assume that CFLs were closed under complement.
  - If L is a CFL then L^C is a CFL
  - Since CFLs are closed under union, $\mathbf{L_1^C \cup L_2^C}$ is a CFL
  - And by our assumption $\mathbf{(L_1^C \cup L_2^C)^C}$ is a CFL
  - But $\mathbf{(L_1^C \cup L_2^C)^C = L_1 \cap L_2}$ which we just showed isn't necessarily a CFL.
  - Contradiction!

Set Difference

- L_1 and L_2 are CFLs. L_1 - L_2 is not necessarily a CFL

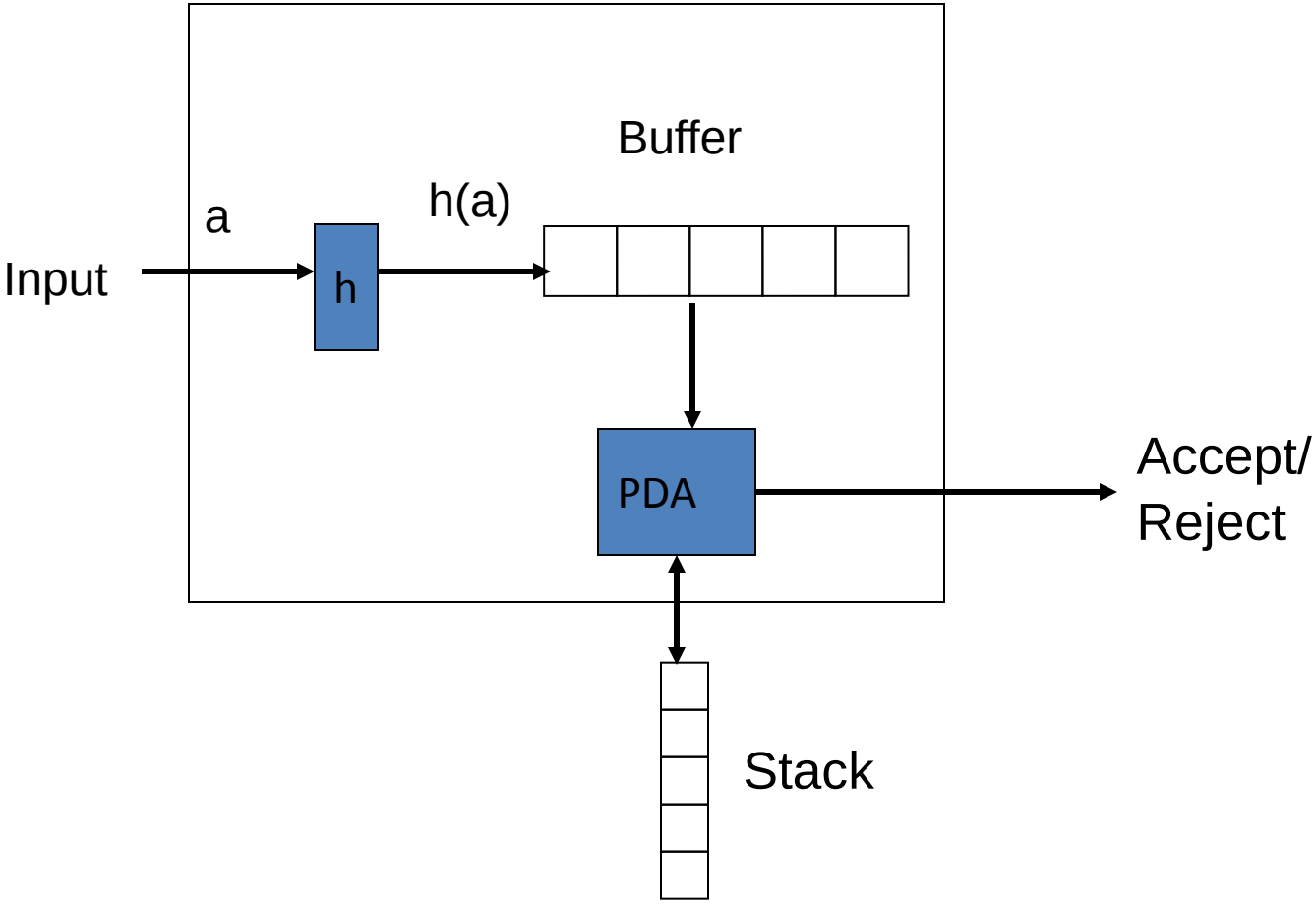
  Proof:

  - L1 = $\sum^*$ - L
  - $\sum^*$ is regular and is also CFL
  - But $\sum^*$ - L = L^C
  - If CFLs were closed under set difference, then $\sum^*$ - L = L^C would always be a CFL.
  - But CFL's are not closed under complementation

Inverse homomorphism

- To recall: If h is a homomorphism, and L is any language, then $h^{-1}(L)$, called an *inverse homomorphism*, is the set of all strings w such that $h(w) \in L$
- The CFL's are closed under inverse homomorphism.
- Theorem: If L is a CFL and h is a homomorphism, then $h^{-1}(L)$ is a CFL

Inverse homomorphism – proof



- After input a is read, h(a) is placed in a buffer.
- Symbols of h(a) are used one at a time and fed to PDA being simulated.
- Only when the buffer is empty does the PDA read another of its input symbol and apply homomorphism to it.

- Suppose h applies to symbols of alphabet Σ and produces strings in T*.
- Let PDA $P = (Q, T, \Gamma, \delta, q_0, Z_0, F)$ that accept CFL L by final state.
- Construct a new PDA $P' = (Q', \Sigma, \Gamma, \delta', (q_0, \varepsilon), Z_0, F \times \{\varepsilon\})$ to simulate language of $h^{-1}(L)$, where
  - Q' is the set of pairs (q, x) such that
    - q is a state in Q
    - x is a suffix of some string h(a) for some input string a in Σ

- δ′ is defined by
  - δ′((q, ε), a, X) = {((q, h(a)),a,X)}
  - If δ(q, b, X) = {(p, γ)} where b∈T or b = ε then δ′((q, bx), ε, X) = {((p, x), γ)}
- The start state of P' is (q0, ε)
- The accepting state of P′ is (q, ε), where q is an accepting state of P.
- $(q_0,h(w),Z_0)$|-*P (p,ε,γ) iff $((q_0,\varepsilon),w,Z_0)$ |-*P′ ((p, ε), ε, γ)
- P accepts h(w) if and only if P′ accepts w, because of the way the accepting states of P′ are defined.
- Thus $L(P')=h^{-1}(L(P))$

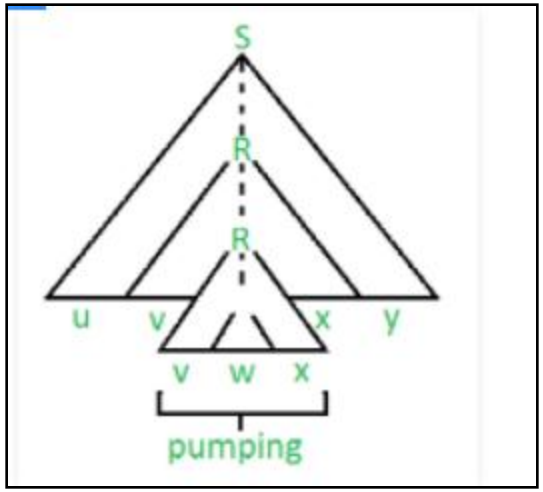
Pumping Lemma

- Pumping Lemma for CFL states that for any Context Free Language L, it is possible to find two substrings that can be ‘pumped’ any number of times and still be in the same language.

- Pumping Lemma is used as a proof for irregularity of a language.
- Thus, if a language is cfl, it always satisfies pumping lemma.
- If there exists at least one string made from pumping which is not in L, then L is surely not regular.
- The opposite of this may not always be true.
- That is, if Pumping Lemma holds, it does not mean that the language is cfl.

- For any language L, we break its strings into five parts and pump second and fourth substring.
- Pumping Lemma, here also, is used as a tool to prove that a language is not CFL.
- Because, if any one string does not satisfy its conditions, then the language is not CFL.

- Thus, if L is a CFL, there exists an integer n, such that for all x ∈ L with |x| ≥ n, there exists u, v, w, x, y ∈ Σ∗, such that x = uvwxy, and
  (1) |vwx| ≤ n
  (2) |vx| ≥ 1
  (3) for all i ≥ 0: $uv^iwx^iy \in L$



Example Problem

Find out whether the language L = {$x^n y^n z^n$ | n ≥ 1} is context free or not.

Solution

Let **L** is context free. Then, **L** must satisfy pumping lemma.

At first, choose a number **n** of the pumping lemma. Then, take z as $0^n 1^n 2^n$.

Break **z** into **uvwxy,** where

|vwx| ≤ n and vx ≠ ε.

Hence **vwx** cannot involve both 0s and 2s, since the last 0 and the first 2 are at least (n+1) positions apart. There are two cases −

Case 1 − **vwx** has no 2s. Then **vx** has only 0s and 1s. Then **uwy**, which would have to be in **L**, has **n** 2s, but fewer than **n** 0s or 1s.

Case 2 − **vwx** has no 0s.

Here contradiction occurs.

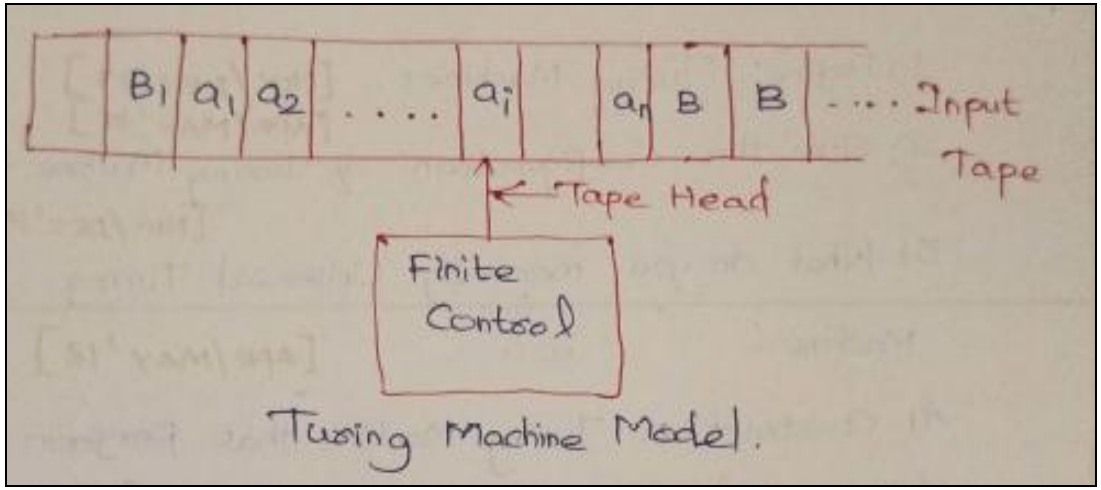
Hence, **L** is not a context-free language.

Applications of Pumping Lemma

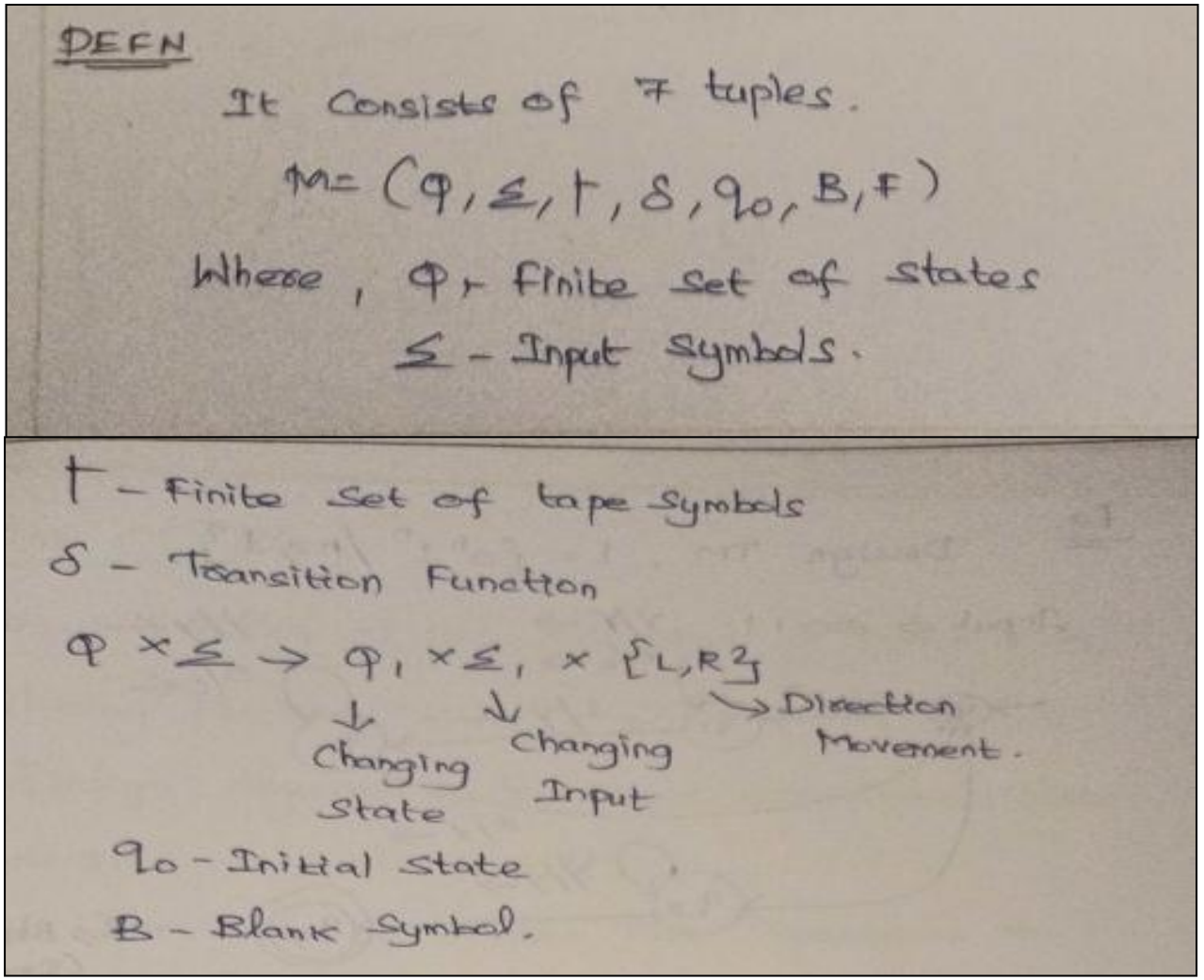
- Pumping Lemma is to be applied to show that certain languages are not regular.
- It should never be used to show a language is regular.
- If **L** is regular, it satisfies Pumping Lemma.
- If **L** does not satisfy Pumping Lemma, it is non-regular.

Turing Machines

A **Turing machine** is a mathematical model of computation that defines an abstract **machine**, which manipulates symbols on a strip of tape according to a table of rules.



Definition of Turing Machines



INSTANTANEOUS DESCRIPTION OF TM

Stmt

$x_1, x_2, x_3 \ldots x_{i-1}, q_0, x_i, x_{i+1} \ldots$

Condition

$$\delta(q_0, x_i) = (P, Y, L)$$

After Condition $x_1, x_2, x_3 \ldots x_{i-1}, q_0, x_i, x_{i+1} \ldots x_n$

$x_1, x_2, x_3 \ldots x_{i-1}, P, Y, x_{i+1} \ldots x_n$ (L)

$x_1, x_2, x_3 \ldots x_{i-2}, P, x_{i-1}, Y, x_{i+1} \ldots x_n$

Condition

$$\delta(q_0, x_i) = (P, Y, R)$$

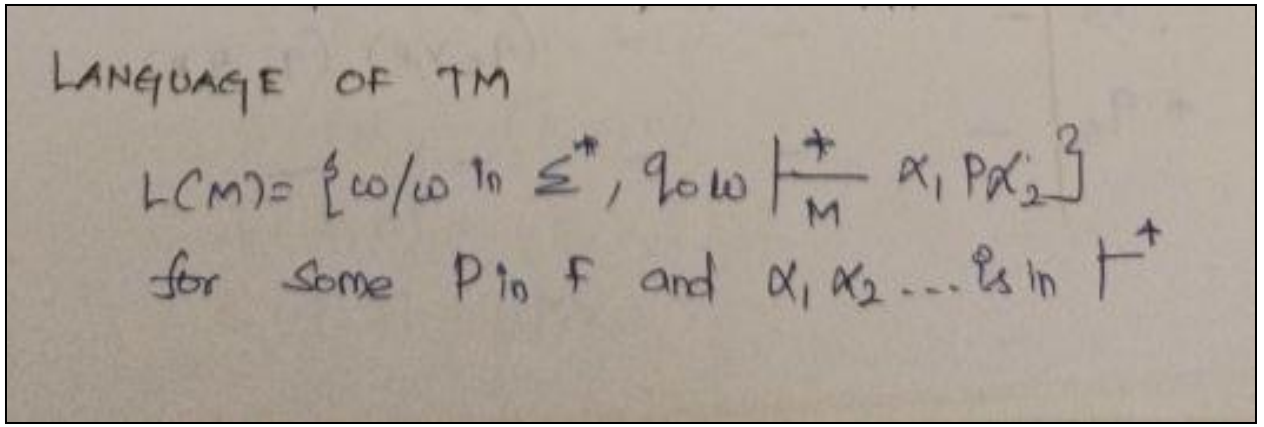
After Condition

$x_1, x_2 \ldots x_{i-1}, P, Y, x_{i+1} \ldots x_n$

$x_1, x_2 \ldots x_{i-1}, P, Y, x_{i+1} \ldots x_n$ (R)

$x_1, x_2 \ldots x_{i-1}, Y, P, x_{i+1} \ldots x_n$

Language of TM



HALTING OF TM

Input string => Not Accepted => Never Halts

Turing Machine => Halts => Accepting state.

Design Algorithm => to check acceptance State.

TRANSITION DIAGRAM OF TM

The Set of Nodes that represents the States of TM.

An Arc from one state to another state is labelled by one or more items of the form,

X/Y D

Where, X, Y → Tape Symbols

D → Direction of Move

(Left / Right)

Left => (⟵)

Right => (⟶)

Example Problems

1) Consider the Transition diagram for the TM is,

$M = (\{P, q\}, \{0, 1\}, \{X, Y\}, \delta, P, B, \{q\})$

And the transition functions are,

$\delta(P, 0) = (q, Y, L)$

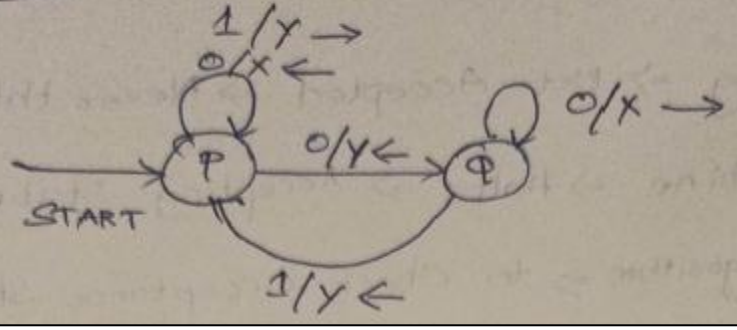
$\delta(P, 0) = (P, X, L)$

$\delta(P, 2) = (P, Y, R)$

$\delta(q, 0) = (q, X, R)$

$\delta(q, 1) = (P, Y, L)$

Solution



2) Design Turing Machine for proper addition

(or)

Design Turing Machine to Compute $f(m+n)=m+n$ for all $m,n \geq 0$

Solution

Assume 2+3=5

Unary Number System

Steps

11 + 111 = 11111

11+111Δ → Move Right side

11+111Δ → Right side (2nd Move)

11+111Δ → + Convert to 1 (Move Right side)

11+111Δ → 1 Move (Right side)

11111Δ → Right side Move

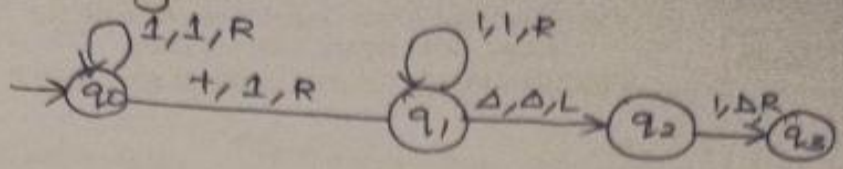
Δ ⇒ End state

11111Δ → Right side Move

11111Δ → Right side Move

111111Δ → Δ Converted to Δ & Move Left side

Transition Diagram



Transition Table

	1	+	Δ
q_0	$(q_0,1,R)$	$(q_1,1,R)$	–
q_1	$(q_1,1,R)$		(q_2,Δ,L)
q_2	(q_3,Δ,R)		
q_3	–	–	–

∴ Turing Machine

$$M=\{(\{q_0,q_1,q_2,q_3\},\{1,+,\Delta\},q_0,q_3,\delta,\{1,\Delta\},\Delta\}$$

Programming Techniques for TM's

- TM's may be used as a computer as well, not just a language recognizer.
- **Example** Design a TM to compute a function called *monus*, or *proper subtraction* defined by

$$m \quad n = m - n \qquad \text{if } m \geq n;$$
$$= 0 \qquad \text{if } m < n.$$

Programming Techniques for TM's

- **Example 8.4** (cont'd)
- Assume input integers *m* and *n* are put on the input tape separated by a 1 as 0^m10^n
- The TM is $M = (\{q_0, q_1, \ldots, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B)$.
- *No final state is needed.*

Programming Techniques for TM's

Example (cont'd)

– *M* conducts the following computation steps:

1. find its leftmost 0 and replaces it by a blank;
2. move right, and look for a 1;
3. after finding a 1, move right continuously
4. after finding a 0, replace it by a 1;
5. move left until finding a blank, & then move one cell to the right to get a 0;
6. repeat the above process.

Programming Techniques for TM's

- **Example**
  - $q_0\underline{0}010 \Rightarrow_1 Bq_1\underline{0}10 \Rightarrow_3 B0q_1\underline{1}0 \Rightarrow_4 B01q_2\underline{0} \Rightarrow_5 B0q_3\underline{1}1 \Rightarrow_9$
    $Bq_3\underline{0}11 \Rightarrow_8 q_3\underline{B}011 \Rightarrow_{10} Bq_0\underline{0}11 \Rightarrow_1 BBq_1\underline{1}1 \Rightarrow_4 BB1q_2\underline{1} \Rightarrow_6$
    $BB11q_2\underline{B} \Rightarrow_7 BB1q_4\underline{1} \Rightarrow_{12} BBq_4\underline{1}B \Rightarrow_{12} Bq_4\underline{B}BB \Rightarrow_{13} B0q_6\underline{B}B$
    halt!
  - $q_0\underline{0}100 \Rightarrow Bq_1\underline{1}00 \Rightarrow B1q_2\underline{0}0 \Rightarrow Bq_3\underline{1}10 \Rightarrow q_3\underline{B}110 \Rightarrow$
    $Bq_0\underline{1}10 \Rightarrow BBq_5\underline{1}0 \Rightarrow BBBq_5\underline{0} \Rightarrow BBBBq_5\underline{B} \Rightarrow BBBBBq_6$
    halt!

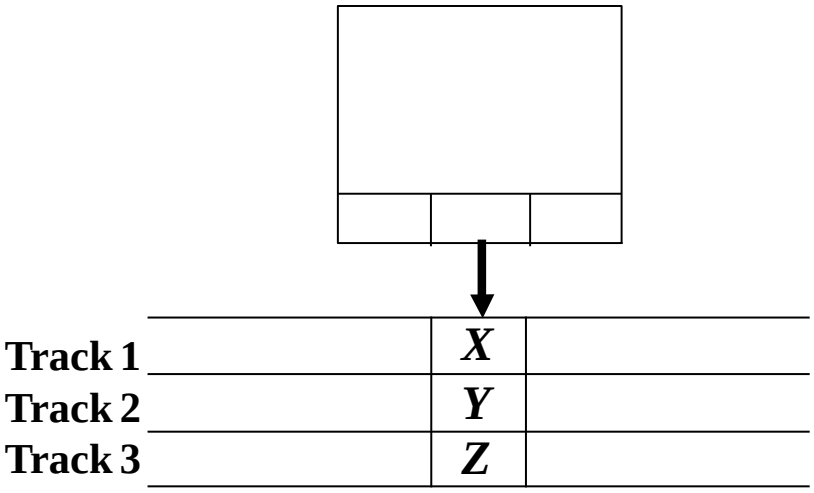
Programming Techniques for TM's

- **Storage in the State**
  - **Technique**:

    use the finite control of a TM to hold a finite amount of data, in addition to the state (which represents a position in a TM "program").
  - Method:

    think of the state as [q, A, B, C], for example, when think of the finite control to hold three data elements A, B, and C. See the figure in the next page

Programming Techniques for TM's



A TM viewed as having finite control storage and multiple tracks.

Programming Techniques for TM's

- **Multiple Tracks**
  - We may think the tape of a TM as composed of several tracks.
  - For example, if there are three tracks, we may use the tape symbol [*X*, *Y*, *Z*] (like that in Figure 8.13).
  - **Example 8.7** --- see the textbook. The TM recognizes the non-CFL language

    $$L = \{wcw \mid w \text{ is in } (\mathbf{0} + \mathbf{1})^+\}.$$

  - Why does not the power of the TM increase in this way?

    Answer: just a kind of *tape symbol labeling*.

Programming Techniques for TM's

- **Subroutines**
  - The concept of subroutine may also be implemented for a TM.
  - For details, see the textbook.
  - **Example 8.8** --- design a TM to perform multiplication on the tape in a way of transformation as follows:

$$0^m 1 0^n 1 \Rightarrow 0^{mn}$$

    For details, see the textbook.

Extensions to the Basic TM

- Extended TM's to be studied:
  - Multitape Turing machine
  - Nondeterministic Turing machine
- The above extensions make no increase of the original TM's power, but make TM's easier to use:
  - Multitape TM --- useful for simulating real computers
  - Nondeterministic TM --- making TM programming easier.

Extensions to the Basic TM

- Multitape TM's

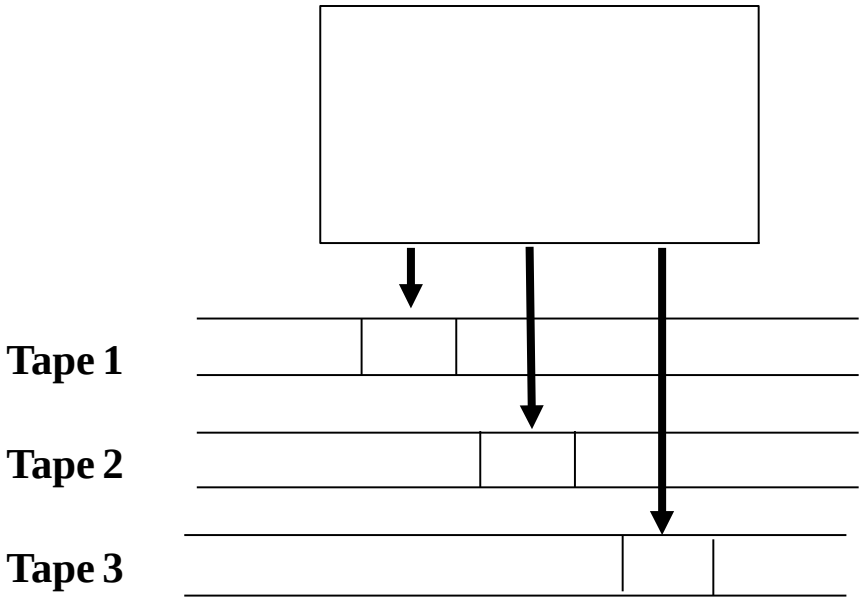


Figure 8.16. A multitape TM.

Extensions to the Basic TM

- Multitape TM's
  - Initially,
    - the input string is placed on the 1st tape;
    - the other tapes hold all blanks;
    - the finite control is in its initial state;
    - the head of the 1st tape is at the left end of the input;
    - the tape heads of all other tapes are at arbitrary positions.
  - A move consists of the following steps:
    - the finite control enters a new state;
    - on each tape, a symbol is written;
    - each tape head moves left or right, or *stationary*.

Extensions to the Basic TM

- Nondeterministic TM's
  - A nondeterministic TM (NTM) has multiple choices of next moves, i.e.,
    $\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$.
  - The NTM is not any 'powerful' than a deterministic TM (DTM), as said by the following theorem.
  - **Theorem 8.11**

    If M_N is NTM, then there is a DTM M_D such that $L(M_N) = L(M_D)$. (for proof, see the textbook)

8.4 Extensions to the Basic TM

- Nondeterministic TM's
  - The equivalent DTM constructed for a NTM in the last theorem may take exponentially more time than the DTM.
  - It is unknown whether or not this *exponential slowdown* is necessary!
  - More investigation will be done in Chapter 10.

Restricted TM's

- Restricted TM's to be studied:
  - the tape is infinite only to the right, and the blank cannot be used as a replacement symbol;
  - the tapes are only used as stacks ("stack machines");
  - the stacks are used as counters only ("counter machines").
- The above restrictions make no decrease of the original TM's power, but are useful for theorem proving.
- Undecidability of the TM also applies to these restricted TM's.

Restricted TM's

- Multistack Machines
  - Multistack machines, which are restricted versions of TM's, may be regarded as extensions of pushdown automata (PDA's).
  - Actually, a PDA with *two* stacks has the same computation power as the TM.

Restricted TM's

- Counter Machines
  - There are two ways to think of a counter machine.
  - Way 1: as a multistack machine with each stack replaced by a counter *regarded to be on a tape of a TM*.
    - A counter holds any nonnegative integer.
    - The machine can only distinguish zero and nonzero counters.
    - A move conducts the following operations:
      - changing the state;
      - add or subtract 1 from a counter which cannot becomes negative.

Restricted TM's

- The Power of Counter Machines
  - Every language accepted by a one-counter machine is a CFL.
  - Every language accepted by a counter machine (of any number of counters) is recursive enumerable.

Thank You