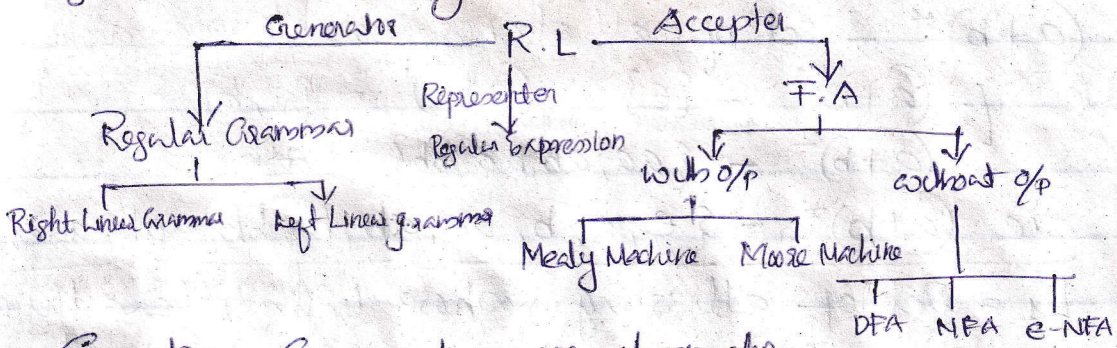
REGULAR EXPRESSIONS & REGULAR GRAMMARS

Languages Accepted by finite Automata are called regular Langu



Generator – Given a language, it generates all the strings in the language. The Grammars which generates R.L is called Regular Grammar.

Regular Expressions : It is a set of characters that represents the search pattern for any string (written in algebraic form)

Regular Expression contains 3 operators

(i) Union +

(ii) Concatenation .

(iii) Kleene closure *

a) ϕ – is R.E. representing $L = \{\}$ – empty set

e – representing the language $L = \{e\}$ – empty string.

$a \in \Sigma$ then and 'a' is the RE ~~is also a R.E~~ $R = \{a\}$ represents the language

These are called primitive Regular Expressions.

b) $r_1 + r_2$ / $r_1 . r_2$ / r_1^* (where r_1 & r_2 are primitive RE)

$L(r_1) \cup L(r_2)$ ← $L(r_1 + r_2)$ → $L(r_1) . L(r_2)$ ← concatenation. $L(r_1 . r_2)$ $L(r_1^*) = (L(R))^*$

then these are R.Es.

c) Using (a) & (b) any number of times is also a R.E.

(17) L_{17} = set of all strings in which no two a's com... toge...

$(b+ab)^*(\epsilon+a)$

or

$(a+\epsilon)(b+ba)^*$

L = {ϵ, b, bb, a, ab, abab, aba...
building blocks are b, or ab or ba
but not a
of ab & ba together we'll get odd c...
some baab
if we use $(b+ab)^*$ = {ϵ, b, bb.., ab...
but all strings are ending in b only
so add another term $(b+ab)^*a$ for ge...
all strings end with a
so $(b+ab)^* + (b+ab)^*a$
ie $(b+ab)^*(\epsilon+a)$

(18) L_{18} no two a's or no two b's come together.

L_{18} = {ϵ, a, b, ab, ba, aba, bab, ...

$(ab)^*a + (ab)^* + b(ab)^* + b(ab)^*$

$= (ab)^*(a+\epsilon) + b(ab)^*(a+\epsilon)$

$= (\epsilon+b)(ab)^*(a+\epsilon)$

or

$(\epsilon+a)(ba)^*(b+\epsilon)$

All classes of languages
strings — starts with — ends with — R.E.

{a, aba, ababa} — a — a — $(ab)^*a$ or ...
{ab, abab, ababab...} — a — b — $(ab)^*$ or ...
{ba, baba, bababa...} — b — a — $(ba)^*$ or b...
{b, bab, babab ...} — b — b — $(ba)^*b$ or ...

Identities of RE

$\phi + R = R + \phi = R$

$\phi . R = R . \phi = \phi$

$\epsilon . R + R . \epsilon = R$

$\epsilon^* = \epsilon$

$\phi^* = \epsilon$

empty set applied with * gives ϵ

$R^+ + \epsilon = R^*$

$\epsilon + RR^* = R^*R + \epsilon = R^*$

$(a+b)^* = (a^*+b^*)^* = (a^* . b^*)^*$

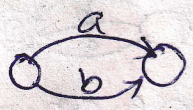
$= (a^*+b)^* = (a+b^*)^*$

$= a^*(ba^*)^* = b^*(ab^*)^*$

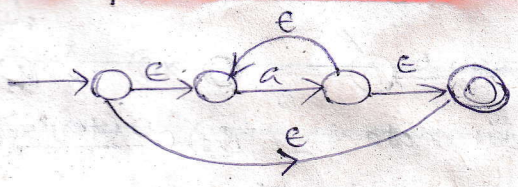
RE to FA

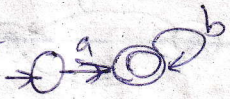
① ϕ: 

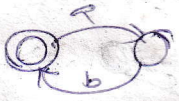
② a: 

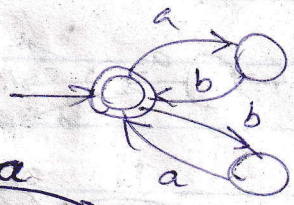
③ a/b 

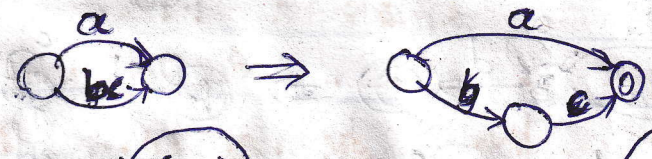
④ a.b ; 

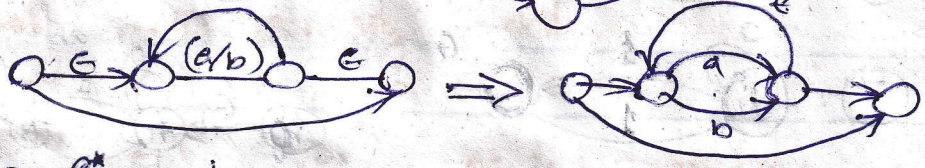
⑤ a^*: 

ab^* 

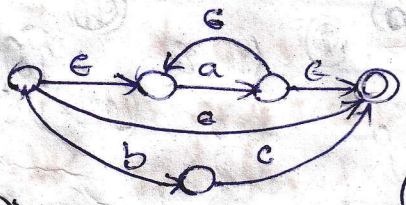
$(ab)^*$: 

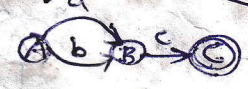
$(ab+ba)^*$ 

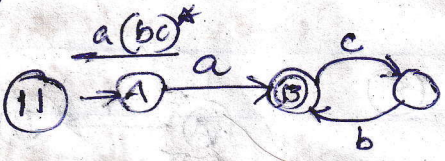
⑥ a+bc 

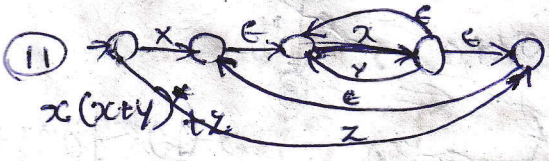
⑦ $(a/b)^*$ 

⑧ a^*bc 

⑨ a^*+bc 

⑩ (a+b)c 

⑪ a(bc)* 

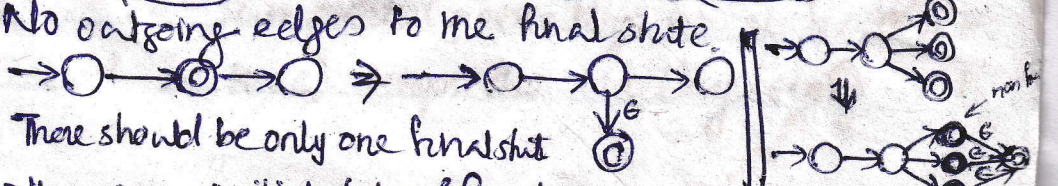
⑪ 

FA to RE

state elimination method

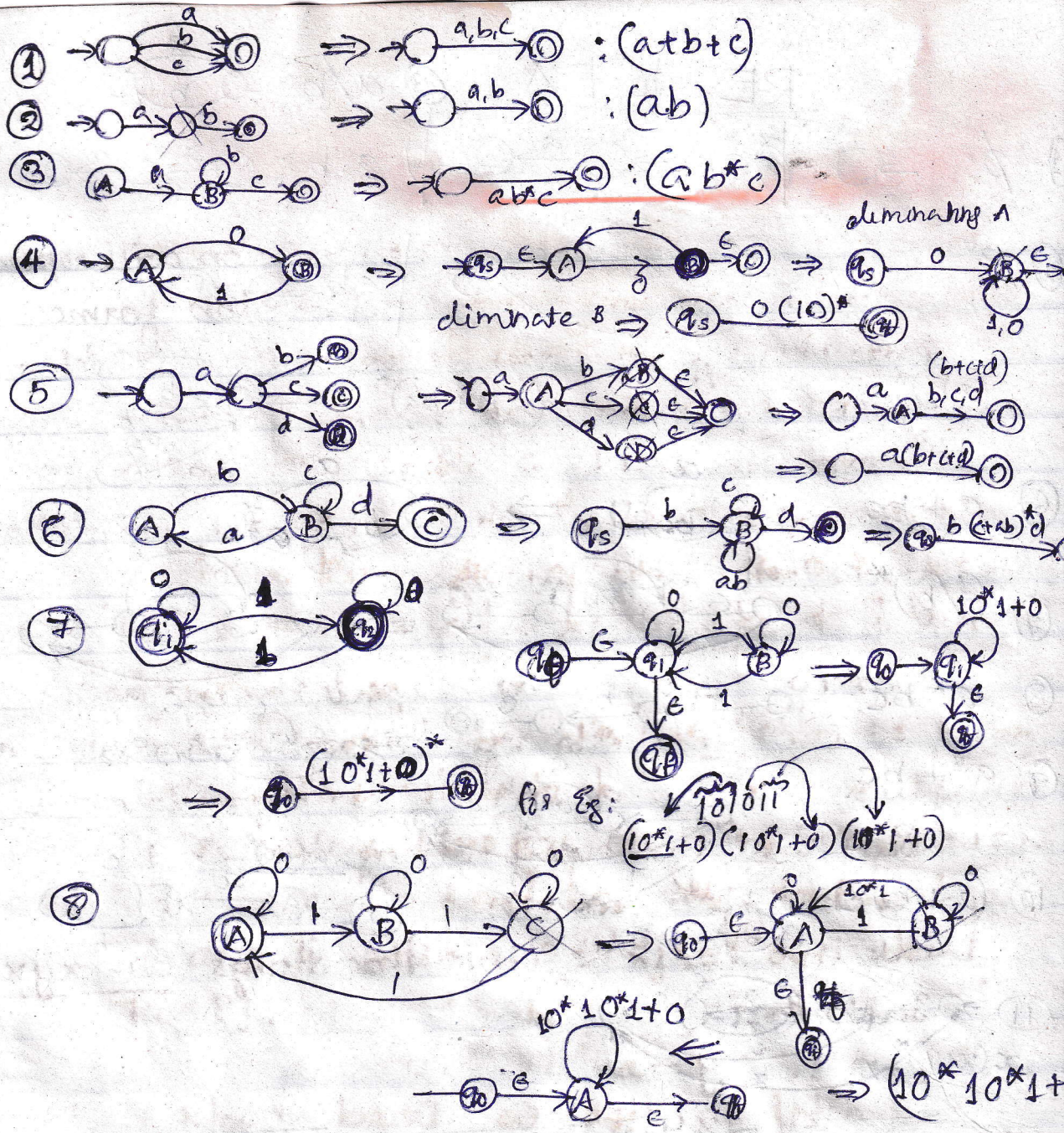
Rules 1. Initial state should not have any incoming edges

2a) No outgoing edges to the final state.



2b) There should be only one final state

3. Other than initial state & final state eliminate all the other states. (All the paths ~~should~~ which where there previously should)


1
a
b
c
a,b,c
: (a+b+c)
2
a
b
a,b
: (a.b)
3
A
a
B
b
c
a b* c
: (a.b* c)
4
0
1
eliminating 1
dimnate B
0 (10)*
5
b
c
d
a
(b+c+d)
b,c,d
6
b
c
a
d
ab
7
0
1
10*1+0
(10*1+0)*
for Eg:
1011
(10*1+0)(10*1+0)(10*1+0)
8
10*1
10*10*1+0
(10*10*1+

PUMPING LEMMA

Every regular expression will represent a regular language and the automaton formed corresponding to a regular expressions is deterministic finite automaton. The regular languages are accepted by DFA's, by NFA's & by ϵ-NFA's. "Pumping Lemma" is a necessary condition for a language to be regular.

Theorem: Let 'L' be a regular language.

'n' - No. of states of DFA which accept L (n depends on L).

Such that for every string 'w' in L

such that $|w| \geq n$,

we can break 'w' into three strings $w = xyz$

such that

1. $y \neq \epsilon$
2. $|xy| \leq n$
3. For all $k \geq 0$, the string xy^kz is also in L.

That is, we can always find a non empty string 'y' not too far from the begining of w that can be pumped. (repeating no. of times ie $k>0$ or deleted ie $k=0$) keeps the resulting string in 'L'.

Proof: Suppose L is regular. Then $L = L(A)$ for some DFA 'A'. Suppose A has 'n' states. Consider any string 'w' of length 'n' or more $w = a_1 a_2 \dots a_m$ where $m \geq n$.

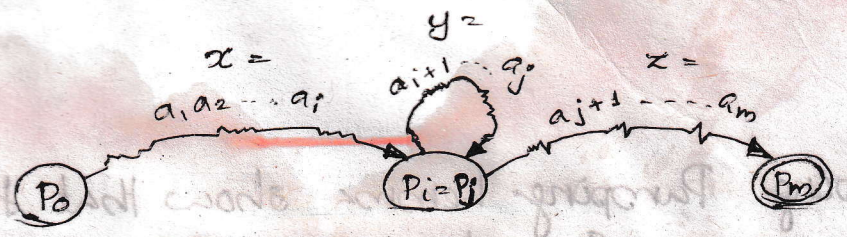
Now define state P_i

$P_i = \hat{\delta}(P_0, a_1 a_2 \dots a_i)$ where δ is transition function of DFA & P_0 is the start state. 'P_i' is the state after reading first 'i' i/p symbols.

For $i = 0, 1, \dots n$ (ie for $n+1$ i/p symbols) It is not possible to have '$n+1$' different P_i's, since there are only 'n' diff. states. Thus we can find two different integers i and j with $0 \leq i < j \leq n$ such that $P_i = P_j$.

Now break $w = xyz$ as follows

1. $x = a_1 a_2 \dots a_i$ — state P_i
2. $y = a_{i+1} a_{i+2} \dots a_j$ — state $P_i = P_j$
3. $z = a_{j+1} a_{j+2} \dots a_m$ — state P_m



In this automata x takes to Pi once 'y' takes from Pi back to Pi (since Pi=Pj) and 'z' is balance of w.

[Every string longer than the no. of states must cause a state to repeat]

For the I/P xy^kz for $k \geq 0$

→ If $k=0$ Then automata starts from P0 to Pi on I/P x and this is also Pj; so on receiving 'z' it goes to accepting state. Thus A accepts xz

→ If $k>0$ automata goes from P0 to Pi on input x and circles from Pi to Pi 'k' times on I/P y^k and then goes to accepting state on reading 'z'.

Thus for any $k \geq 0$, xy^kz also accepted by finite automata ie xy^kz is also in L.

1. Using Pumping Lemma show that the Language $L = \{ww \mid w \in \{0,1\}^*\}$ is not regular.

Let 'n' be a +ve integer

Let $w = 0^n 1$ & $u = ww = 0^n 1 0^n 1 \in L$

$|w| = 2n+2 > n$

Consider the decomposition of 'u' as

$u = xyz$ where $x, y, z \in \{0,1\}^*$ satisfying condition $|y| \geq 1$ and $|xy| \leq n$

For every decomposition of 'u' it is clear that y consists of only 0's

Let $y = 0^i$ for some $i > 0$

Take $k = 0$ in string xy^kz ie xy^0z.

$xy^0z = xz = 0^{n-i}10^n1$ where $n-i < n$

ie $xz \notin L$ so L is not regular.

2. By Ex: take $w = 00111$ $u = ww = \underbrace{00}_{x}\underbrace{111}_{y}\underbrace{00111}_{z}$

$u = xyz$ in xy^kz let $k=2$.

ie $\underbrace{00}_{x}\underbrace{111}_{y^2}\underbrace{00111}_{z}$ ie xy^2z can't be represented in ww form.

00111 00111

$a^{n-1} b a^n$

2. Find out whether $L = \{ w \in \{a,b\} \mid w = w^R \}$ is regular or not?

Let n be a +ve integer $n \geq 1$ and ω be $a^n b a^n$

Consider the decomposition of ω. We observe that y will contain only 'a's (since $|xy| \leq n$)

$\omega = xyz$, $x, y, z \in \{a,b\}^*$ $|xy| \leq n$ & $|y| \geq 1$

hey $y = a^i$

let $k = 2$ in $\omega = xy^k z$

ie $\overline{\omega} = xy^2z$

$= a^{n-k} (a^k)^2 b a^n$

$= a^{n+k} b a^n$

Since $n+k > n$ $\overline{\omega}$ not belong to L

Hence $\overline{\omega} = \overline{\omega}^R$ is not satisfied.

3. Show that the set

$L = \{ 0^{i^2}, i \geq 1 \}$ is not regular.

Let $\omega = 0^{n^2}$ then $|\omega| = n^2 > n$ by P.L.

$\omega = xyz$ with $|xy| \leq n$ & $|y| > 0$.

Consider xy^2z

$|xy^2z| = |x| + 2|y| + |z| > |x| + |y| + |z|$

This means $|xyz| = n^2$

ie $|x| + |y| + |z| < |xy^2z|$ as $|xy| \leq n, |y| \leq n$

$\therefore |xy^2z| = |x| + 2|y| + |z| \leq n^2 + n$

ie $n^2 < |xy^2z| < n^2 + n < n^2 + n + n + 1$

$n^2 < |xy^2z| < (n+1)^2$

Hence $|xy^2z|$ should lie b/w n^2 & $(n+1)^2$ which is not a perfect square. So $xy^2z \notin L$

4. Show that the ~~regular~~ set
$L = \{ 1^p / p \text{ is a prime} \}$ is not regular.

Let p be a prime number greater than n
Let $w = 1^p$ ~~$|w| = p$~~. By Pumping Lemma
$w = xyz$ such $|xy| \le n$ & $|y| > 0$
Let $y = 1^m$ for some $m \ge 1$

Let $i = p+1$ then $|xy^iz| = |xyz| + |y^{i-1}|$ ($i = p+1$; $|y| = 1$; $p+1-1$)
$= p + (i-1)m$
$= p + pm$
$= p(1+m)$

$|xz| = m$ a prime no.
for any $0 \le m \le$ ~~∞~~ $|xy^mz| = m + |y|m$
$= m(1+|y|)$ is not a prime

By Pumping Lemma
xy^iz ~~is~~ has $p(1+m)$ length.
Since $p(1+m)$ is not a prime number,
$xy^iz \notin L$. This is a contradiction
Thus L is not regular.

5 Show that the language "Any no. of zeros followed by same no. of ones" is not a regular language.

Let $w = 0^n 1^n$

$|w| = 2n > n$ by pumping lemma.

Consider the decomposition of w as

$w = xyz$ where $x, y, z \in \{0,1\}^*$ and

$|xy| \leq n$ & $|y| \geq 1$

In such every decomposition of 'w' it is clear that xy contains only 0's.

hey $y = 0^i$ for some $i > 0$.

In string $xy^k z$, take $k = 0$ so that xz should be accepted if it is a regular set.

ie $xy^k z = xy^0 z = xz$

$xz = 0^{n-i} 1^n$

Since xz is not in L; it is a contradiction and L is not a R.L.

6. Let $L = \{a^{n!} \mid n > 2\}$ Show that L is not regular.

Let $w = a^{n!}$ & $|w| \geq n$ by P.L.

Decomposition of $w = xyz$ where $x, y, z \in \Sigma^*$ and $|xy| \leq n$ and $|y| \geq 1$.

$(n-1)! \times n$

Let $|y| = i < n$

So $w = xy^k z$ and $k=0$

then $xy^0 z = xz = a^{n!-i}$ Since $n>2$ and $i<n$, $n! - i > (n-1)!$

So $xz \notin L$ Hence L is not regular.

7. Let $L = \{x \in \{0,1\}^* \mid x = ww^R, w \in \Sigma^*\}$

Show that L is not regular.

Let $w = 0^n 1^n$ $|x| = 4n > n$ by P.L.

Let us take the split of x as

$x = uvw$, where $|uv| \leq n$ and $|v| \geq 1$

For every decomposition of x; it is clear that v contains only 0's.

Let $v = 0^i$ for some $i > 0$.

ie $x = \underbrace{0^{n-i}}_{u} \underbrace{0^i}_{v} \underbrace{1^n 0^n}_{w} = uv^k w$

Let us take $k=0$ in $uv^k w$

ie $x = 0^{n-i} 1^n 0^n$

Since $i \geq 1$ (because $|v| \geq 1$) $n-i < n$

ie x has fewer zeroes on the left than on the right. So it can't be of the form ww^R. Hence L is not regular.

8 $L = \{x \in \{a,b\}^* \mid |x|_a \geq |x|_b\}$

Let $\omega = a^{n+1} b^n$

Let $\omega = xyz$ where $x, y, z \in \Sigma^*$, $|xy| \leq n$ & $|y| \geq 1$

From the decomposition it is clear that 'y' contains only a's. Let $i \geq 0$ is an integer where $y = a^i$

ie $xyz = a^{(n+1-i)} a^i b^n$. Since $i > 0$

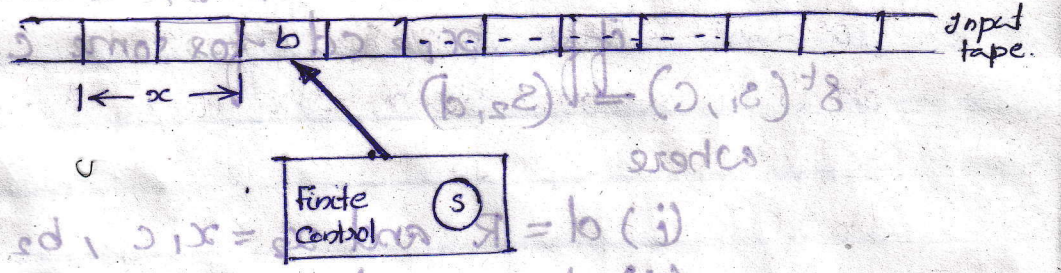
$xy^k z = a^{(n+1)-i} a^i b^n$ Let $k = 0$ then

$xz = a^{n+1-i} b^n$ Since $i \geq 1$ it is not possible to have more a's than b's.

So it is a contradiction and hence L is not regular

TWO WAY FINITE AUTOMATON

A 2-way F.A. is a mathematical model of the type deterministic finite automaton, but with additional feature in it that the tape head is allowed not only to move forward but also backwards on the input tape.



A two way finite automaton M is defined by a quintuple $< S, \Sigma, \delta^t, s, F >$ where S, Σ, s and F have the usual meanings defined for dfa and δ^t is defined as a function from $S \times \Sigma$ into $S \times \{L, R\}$, where L & R denote the directions of tape head movement.

Configuration of a 2-way F.A $< s, x, b >$ where s denotes the current state, b is the symbol at which the tape head is currently

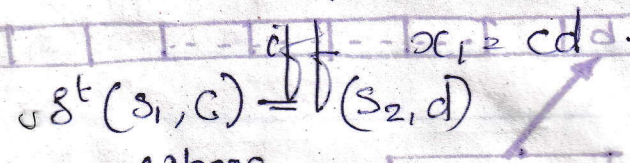
positioned and x_1 is the string on the left side of the symbol 'b' on the tape.

Moves in 2-way finite Automaton

The transitions of a two way finite automaton can be defined as

$$\langle s_1, x_1, b_1 \rangle \vdash_M \langle s_2, x_2, b_2 \rangle$$

if $x_1 = cd$ for some $c \in \Sigma$ and $\delta^t(s_1, c) = (s_2, d)$



where

(i) $d = R$ and $x_2 = x_1 c$, $b_2 = d$ or

(ii) $d = L$ and $x_1 = ye$ for some $y \in \Sigma^*$ & $x_2 = eb_1$

when $b = \lambda$, the configuration $\langle b, x, \lambda \rangle$ implies that the automaton TM has completed its operation on string x and reached the state 't'.

Generating Grammars

① Ex: $L = \{aa, ab, bb, ba\}$ we can write
$S \to aa/ab/bb/ba$ instead $(a+b)(a+b)$
is RE.
we can write $S \to AA$, $A \to a/b$ ✗
$A \to a/b$ ✓ $S \to aA/bA/a/b$

② $a^n / n \geq 0 = \{a, aa, aaa, \ldots\}$.
$A \to aA/\epsilon$ ✓

③ $(a+b)^*$ ✓ $S \to aS/bS/\epsilon$

④ Atleast 2 symbols.
$(a+b)(a+b)(a+b)^*$
$A \to a/b$, $B \to aB/bB/\epsilon$
$S \to AAB$. $S \to aA/bA$ $A \to a/b/aA/bA$

⑤ At most 2 length. $S \to aA/bA/\epsilon$, $A \to a/b/\epsilon$
$(a+b+\epsilon)(a+b+\epsilon)$
$A \to a/b/\epsilon$
$S \to AA$ $A \to a/b/\epsilon$

⑥ Start with a and end with b
$a(a+b)^*b$ $S \to aA/bB$, $A \to aA/bA/b$
$S \to aAb$ $A \to aA/bA/\epsilon$

⑦ Starting & ending with diff symbol
$S \to aAb/bAa$
$A \to aA/bA/\epsilon$
$a(a+b)^*b + b(a+b)^*a$
$S \to aA/bB$
$A \to aA/bA/b$
$B \to aB/bB/a$

⑧ Starting & ending with same $S \to aA/bB/a/b/\epsilon$, $A \to aA/bA/a$, $B \to aB/bB/b$
$S \to aAa/bAb/a/b/\epsilon$
$A \to aA/bA/\epsilon$

⑨ $a^n b^n / n \geq 1$
$S \to aSb/ab$

⑩ $(\omega\omega^R) \cup (\omega a \omega^R \cup \omega b \omega^R)$ (even length palindrome, odd length palindrome)
$S \to aSa/bSb/a/b/\epsilon$

⑪ Even length strings $S \to aaS/bbS/abS/baS/\epsilon$
$S \to BS/\epsilon$, $B \to AA$, $A \to a/b$
$(a+b)(a+b)^*$ $A \to a/b$

⑫ Odd length strings $S \to aA/bA/$, $A \to aaA/bbA/abA/baA/\epsilon$
$S \to BS/a/b$ $B \to AA$ $A \to a/b$

⑬ $a^n b^m / n, m \geq 1$
$S \to AB$
$A \to aA/a$ $B \to bB/b$

⑭ $a^n b^n c^m / n, m \geq 1$
$S \to AB$.
$A \to aAb/ab$ $B \to cB/c$

⑮ $a^n c^m b^n / n, m \geq 1$
$S \to aSb/aAb$ $A \to cA/c$

⑯ $a^n b^n c^m d^m / n, m \geq 1$
$S \to AB$
$A \to aAb/ab$ $B \to cBd/cD$

⑰ $a^n b^n c^n / n \geq 1$ ✗ It is context sensitive
we can try $S \to aSbc/abc$ ✗

⑱ $a^n b^{2n} / n \geq 1$
$S \to aSbb/abb$

⑲ $a^n b^m c^m d^n$
$A \to bAc/bc$
$S \to aSd/aAd$

⑳ $a^n b^m c^n d^m$ ★ Not context free

㉒ $a^n b^{n+m} c^m / n, m \geq 1$
$\Rightarrow a^n b^n b^m c^m$
$S \to AB$
$A \to aAb/ab$ & $B \to bBc/bc$

㉓ $a^n b^m c^{n+m}$
$\Rightarrow a^n b^m c^m c^n$

㉔ $a^{m+n} b^m c^n \Rightarrow a^m a^n b^m c^n$
$S \to aSc/aAc$
$A \to aAb/ab$

Neither regular nor context free.

$S \to aX/a/bY/b$

$X \to bK$ $\quad Y \to aL$

$K \to aX/a$ $\quad L \to bY/b$

GRAMMAR

A language is said to be FORMAL LANGUAGE if it can be specified by well-defined set of rules of syntax

A Grammar is a quadruple $G = \langle V, T, P, S \rangle$ where

V - Finite set of vbles, also called non-terminals

T - Finite set of constants, also called terminals.

S - Special symbol called the start vble

P - Finite set of productions or rules.

V & T are non empty but disjoint.

Structure of grammar

$G = \langle V, T, P, S \rangle$ where P consists of products of the form $x \to y$ where $x \in (V \cup T)^+$ ie x is some non-null strings of terminals & vbles $y \in (V \cup T)^*$ ie y is some, possibly null, string of terminals and vbles.

Language by a grammar

Let $G = \langle V, T, P, S \rangle$ be a grammar.

We say that string $\omega \in T^*$ is generated by G if

$$S \overset{*}{\underset{G}{\Rightarrow}} \omega .$$

ie Starting from the start symbol S and

after applying a finite number of productions successively it finally leads to w

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

where w_i's are in $(V \cup T)^+$, for $i = 1, 2, \ldots n$.

ie

$L(G) = \{x \in T^* : S \overset{*}{\underset{G}{\Rightarrow}} x\}$ is the lngue generated by G.

The strings $S, w_1, w_2, \ldots w_n$ are called sentential forms of derivations of word 'w'

1. Find the language generated by grammar

$S \to 1S1$, $S \to 11$

$S \to 1S1 \to 11S11 \to 111S111 \to 1^nS1^n \to 1^n11^n \to 1^{2n+2}$

$L(G) = \{x \in T^* : S \overset{*}{\underset{G}{\Rightarrow}} x \mid x$ contains even no. of $1's\}$

2. Find the grammar that generates language

(i) $L = \{0^n1^m : n, m \geq 0\}$

$S \to AB$, $A \to \in$, $A \to 1A$

$B \to \in$, $B \to 0B$

(ii) $L = \{a^nb^n : n > 0 , a, b \in T\}$

$S \to aSb$, $S \to \in$

(iv) Strings start and end with diff. symbols:
$0(0+1)^*1 \mid 1(0+1)^*0$. $S \to 0A1 \mid 1A0$ $A \to 0A \mid 1A \mid \epsilon$

(iii) $L = \{a^n b^{n+1} : n > 0, a, b \in T\}$
$S \to Ab$, $A \to aAb$, $A \to \epsilon$.

CHOMSKY HIERARCHY

Noam Chomsky introduced the classification scheme for the phrase-structured grammars based on their type of production.

1. Type 0 or Unrestricted type
A grammar $G = (V, T, P, S)$ is called type 0 if all productions are of the form $u \to v$ where $u \in (V \cup T)^+$ and $v \in (V \cup T)^*$
language: Recursive enumerable language
Machine: Turing machine.

2. Type 1 or context sensitive language
A grammar $G = (V, T, P, S)$ is said to be type 1 if all productions are of the form $x \to y$ where $x, y \in (V \cup T)^+$ and $|x| \leq |y|$
Language: Context sensitive language
Machine: Linear Bound Automaton

3. Type 2 or Context-Free language
A grammar $G = (V, T, P, S)$ is said to be context free grammar if its production

(single non terminal)

are given as $\alpha A \beta \rightarrow \alpha / \beta$

$A \rightarrow B$ where $A \in V$, $B \in (V \cup T)^*$

Language : Context Free language

Machine : Pushdown Automata.

4. Type 3 or Regular Language Grammar.

A grammar $G = (V, T, P, S)$ is said to be regular grammar if its production are given as

$A \rightarrow xB$ where $A \in V$, $B \in (V \cup \lambda)$, $x \in \Sigma^*$

Language : Regular Language

Machine : Finite state Automaton.

Every type 3 gram is type 2 gram and every type 2 gram is type 1 gram and every type 1 gram is type 0 gram. i.e.

$L_{RL} \subset L_{CFL} \subset L_{CSL} \subset L_0$

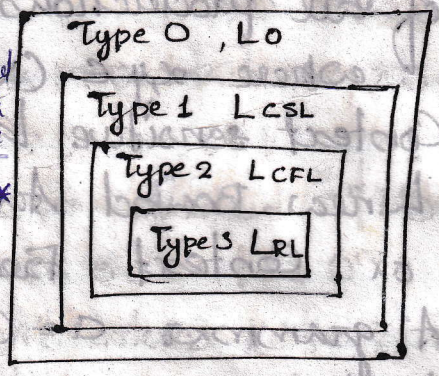
$A, B \in V$, $\alpha, \beta \rightarrow T^*$

Type 3 : $A \rightarrow \alpha B / \beta$ ← right linear, left linear.

$A \rightarrow B\alpha / \beta$

G should be either R.L or L.L should not be a combination. like $A \rightarrow Ba/a$, $B \rightarrow aB/a$

Type 2 : $A \rightarrow \alpha$ $A \in V$, $\alpha \in (V \cup T)^*$



IMP : Converting a F.A to Right Linear Gramm (RLG) is straight forward But inorder to convert it into L.L.G, the procedure is :

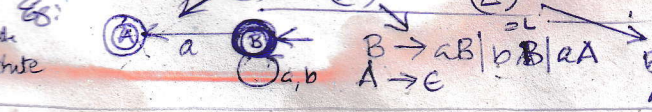
FA $\xrightarrow{R}$ FA $\rightarrow$ RLG $\xrightarrow{R}$ LLG

(L) (L^R) (L^R) (L^R)^R = L

Converting F.A

Final state to unreachable
Initial state to final state

Eg:



$B \rightarrow aB \mid bB \mid aA$

$A \rightarrow \epsilon$

$B \rightarrow Ba/Bb/Aa$

$A \rightarrow \epsilon$

REGULAR GRAMMAR

A grammar $G = (V, T, P, S)$ is said to be regular grammar if it is either right linear grammar or left linear grammar. ie its productions are of the form A

$A \rightarrow xB$

$A \rightarrow x$

or

$A \rightarrow Bx$

$A \rightarrow x$

($A, B \in V$ and $x \in T^*$)

1. Describe the language for the following regular grammars

(a) $S \rightarrow 0X/1S \quad X \rightarrow 1X/1$

ie

$S \rightarrow 0X \rightarrow 01$

or $S \rightarrow 0X \rightarrow 01X \rightarrow 011X \ldots 01^{n-1}X = 01^n$

or $S \rightarrow 1S \rightarrow 11S \rightarrow 111^mS \rightarrow 1^mS \rightarrow 1^m01^n$

$\therefore S \Rightarrow 1^m01^n$ where $m \geq 0 \quad n \geq 1$

So $L(G) = \{1^m01^n \mid m \geq 0, n \geq 1\}$ is a set of strings over $\{0,1\}$ containing exactly one '0' ending with at least one 1.

b) $S \to aS \quad S \to bX \quad X \to bX \mid b$

$S \to aS \to aaS \to \cdots \to a^n bX \to a^n b^m$

$L(G) = \{ a^n b^m \mid n \geq 0 \; m \geq 2 \; m, n \not\to \geq 1 \}$.

c) $S \to a \mid aX, \quad X \to bY \quad Y \to aX \mid a$

Sol: $S \to a$.

$S \to aX \to abY$

$\to abaX \to ababY \to ababa$

ie $S \Rightarrow aba\,ba\,ba$

$S \overset{*}{\Rightarrow} a(ba)^n$

$L(G) = \{ w = a(aba)^n \mid n \geq 0 \}$.

d] $S \to 0A \mid 1S \mid 1 \mid 0 \quad A \to 1A \mid 1S \mid 1$

$S \to 0A \to 01A \to 010A \to 0101S \to 01011 \ldots$

ie there is no chance of any consecutive zeros.

$L(G) = \{ w, w \in (0,1)^* \mid w$ contains no consecutive 0's$\}$.

e] $S \to 0S1 \mid 0A1 \quad A \to 1A \mid 1 \quad L = \{0^m 1^n \mid n > m > 1\}$.

f] $S \to 0S1 \mid 0A1 \quad A \to 10 \mid 10 \quad \therefore L = \{0^n 1^m 0^m 1^n \mid m, n \geq 1\}$

but in context sensitive $aA \to aa$ means A can be replaced by a wherever it is preceded by a.

$A \to aAb$ without considering the context of A (i.e whether A is preceded by somebody (or succeeded by) in) A can be substituted

CONTEXT - FREE LANGUAGES

CFG's are used in a number of applications such as programming languages, parser design, lexical analysis, string matching algorithms, and describing document formats used in XML.

A grammar $G = \langle V, T, P, S \rangle$ is said to be in CFG if rules in production set P are in the form

$$A \to x, \quad A \in V \quad x \in (V \cup T)^*$$

A language 'L' is said to be in context-free language if there exists a context-free grammar 'G' such that $L(G) = L$.

Generate CFG for following Languages.

1. $L_1 = \{ a^n b^n \mid n \text{ is a +ve integer} \}$

Sol: $G = (V, T, P, S)$ $V = \{S\}$, $T = \{a, b\}$

$P = \{ S \to aSb \mid ab \}$

2. $L_2 = \{ w w^R \mid w \in \{a, b\}^+ \}$

$G_2 = \{ V_2, T_2, P_2, S_2 \}$ L_2 : $V_2 = \{S\}$, $T = \{a, b\}$

$P = \{ S \to asa \mid bsb \mid aa \mid bb \}$.

3. $L = \{ a^{2i} b^{3i} \mid i \geq 1 \}$

$S \rightarrow aaSbbb \mid aabbb$

4. $L = \{ a^i b^j c^i \mid i, j \geq 1 \}$

$S \rightarrow aBc \mid aSc \quad , \quad B \rightarrow b \mid bB$

5. $L = \{ a^i b^i c^j \mid i, j \geq 1 \}$

$S \rightarrow DC \qquad D \rightarrow aDb \mid ab \qquad C \rightarrow c \mid cC$

6. $L = \{ a^i b^i a^j b^j \mid i, j \geq 1 \}$

$S \rightarrow CC \qquad C \rightarrow ab \mid aCb$

7. $L = \{ a^n b^m \mid n \neq m \, , \, n, m \geq 0 \}$

$S \rightarrow aSb \mid aA \mid bB \qquad A \rightarrow aA \mid a \mid \epsilon \qquad B \rightarrow bB \mid b \mid \epsilon$

8. $L = \{ a^n b^m c^{n+m} \mid m, n \geq 1 \}$

~~$S \rightarrow aSB$~~

$S \rightarrow asc \mid ac \mid bc \mid bAc$

$A \rightarrow bAc \mid bc$

9. $L = \{ x \in \{0,1\}^* \mid x = x^R \}$

$S \rightarrow 0S0 \mid 1S1 \mid 1 \mid 0 \mid \epsilon$

Derivation Tree OR Parse tree

A parse tree is used to represent the derivations in CFG.

Let $G = \langle V, T, P, S \rangle$ be a CFG, a parse tree is defined to be a tree satisfies following conditions

1. Every node carries a label from V. The topmost node ie root is labelled as 'S'.
2. The bottom most nodes nodes, called leaves are labelled from $T \cup \{\epsilon\}$
3. If an interior node is labelled $A \in V$, and its childrens are labelled as $X_1, X_2 \cdots X_n$ from left to right then $A \to X_1 X_2 \cdots X_n \in P$
4. If a leaf is labelled ϵ then it must be only child of its parent.

1. Let $G = \langle \{S\}, \{0,1\}, \{S \to 0S0 \mid 1S1 \mid \epsilon\}, S \rangle$ Then draw the derivation tree for the derivation

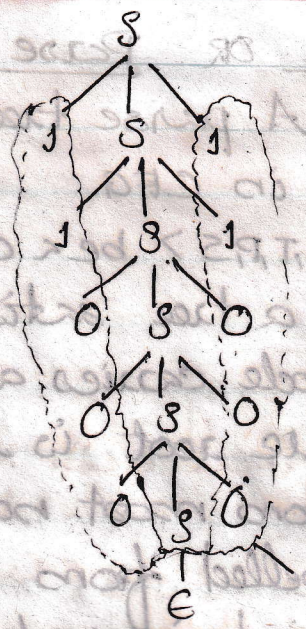
$S \xrightarrow{*} 1100000011$

$S \to 1S1 \to 11S11 \to 1103011 \to 1100S001$

$\to 1100030001 11 \to 11000\epsilon00011$

$\to 1100000011$

YIELD: Yield of a tree is the final string obtained by reading the leaves from left to right ignoring ϵ's



Yield : 1100000011

2] Show the parse tree for string aabbbb a
grammar $S \to AB \mid \varepsilon$ $A \to aB$ $B \to Sb$

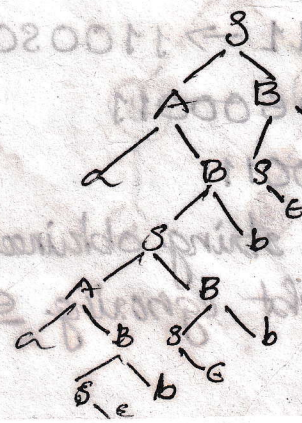
$S \to AB \to aB$

$S \to AB \to aBB \to aSbB \to aABbB$

$\to aaBBbB \to aaSbBbB$

$\to aabBbB \to aabSbbB \to aabbbB$

$\to aabbbSb \to aabbbb.$



AMBIGUITY

A grammar 'G' accepts strings that have two or more distinct derivations or parse trees called ambiguous grammar.

Formal definition

A grammar 'G' is said to be ambiguous if there exist some string of language generated by 'G' which has two or more leftmost or rightmost derivations (hence consequently many two or more distinct parse trees

$S \to S+S/S*S/a/b$

$a+a*b$ [$S \to S+S \to a+S \to a+S*S \to a+a*S \to a+a*b$]

$a+a*b$ [$S \to S*S \to S+S*S \to a+S*S \to a+a*S \to a+a*b$]

1. Show that the following grammars are ambiguous

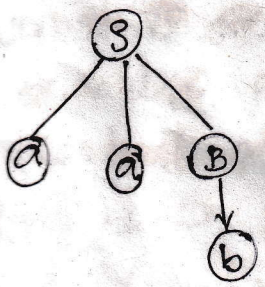
(i) $S \to AB \mid aaB$ $\quad A \to a \mid Aa \quad B \to b$

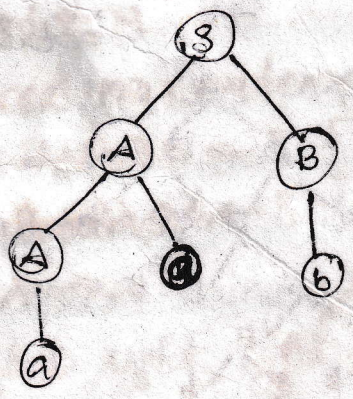
Consider the string $\omega = aab \in L(G)$

$S \Rightarrow aaB \to aab$

$S \Rightarrow AB$

$\Rightarrow AaB \Rightarrow aaB \Rightarrow aab$





(ii) $S \rightarrow aSbS \mid bSaS \mid \epsilon$

let $\omega = abab \in L(G)$

$S \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSbS \Rightarrow aba$

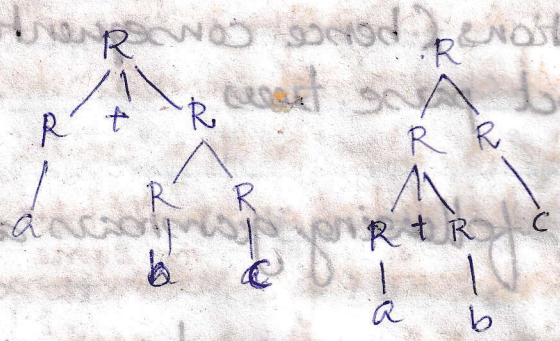
$\Rightarrow abab$

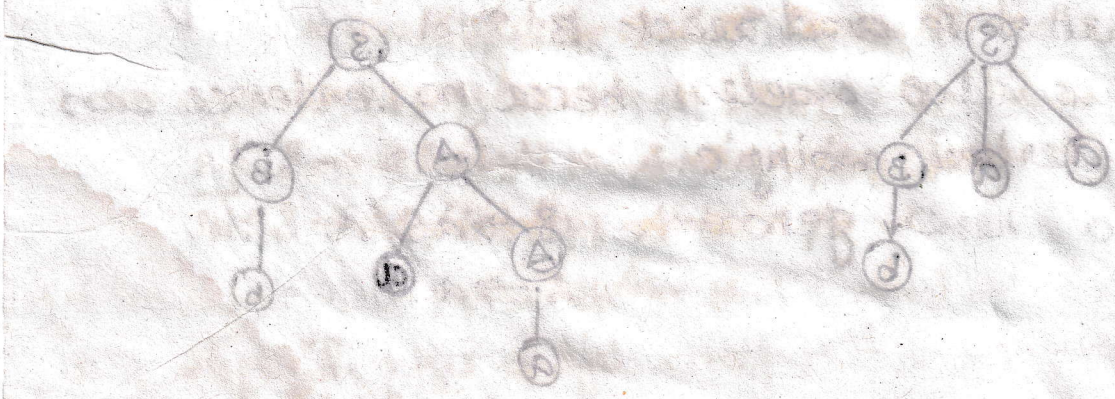
$S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow abaSbS \Rightarrow aba$

$\Rightarrow abab$

(iii) $R \rightarrow R+R \mid RR \mid R^* \mid a \mid b \mid c$ ← Examples of R.E

$a+bc$





Simplification of CFG

In CFG we may not use all symbols for deriving a sentence. So CFG can be simplified by eliminating the productions which are not useful.

ELIMINATION of Useless symbols

A vble in grmr is said to be useless if there is no way of getting a terminal string from it and it can't be reached from the starting symbol.

If X is useful then there will be a derivation $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ for some where $\alpha, \beta, \& w$ where in 'w' is in T^*

1. Eliminate useless symbol from the grammar.

$S \to aS/A/C \quad A \to a \quad B \to ab \quad C \to aCb.$

- B is useless since no string is derived from start state and reach B.
- C is also useless becoz no sentence can be derived using 'C'.

So new gmr: $S \to aS/A$

$A \to a.$

Eliminating ε - productions

If an empty string belongs to a lang
we can eliminate ε from all productions save for
the single production $A \to \epsilon$. In this case
we can eliminate any occurance of A
from right hand side of production.

1] $S \to ABaC$, $A \to BC$, $B \to b/\epsilon$, $C \to D/\epsilon$, D

Sol: Since B & C could goto ε, A could
also goto ε, Look for all places where A,
and C are used on the right hand side of the
production, generate all possible combination
where A, B or C could go to ε.

ie $S \to BaC/AaC/ABa/aC/Ba/Aa/a/ABa$
$A \to BC/B/C$
$B \to b$ $C \to D$ $D \to d$.

2] $S \to aS/bA/$ $A \to aA/\epsilon$

Sol: $S \to aS/bA/b/a$ $A \to aA/a$

Eliminate ε-productions & then useless symbols for

S → AaB/aaB A → ε B → bbA/G

Since A & B could go to ε ....

S → AaB/aB/Aa/a/aaB/aa

B → bbA/bb/b

Then eliminating useless symbols

There is no productions for 'A' so new grammar will be

S → aaB/a/aa/aB

B → bb

ELIMINATING UNIT PRODUCTIONS

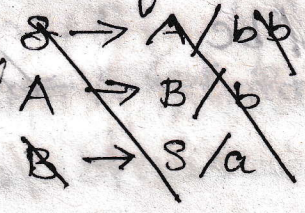
General form of unit production: A → B

A, B ∈ V

Example 1: if A → B B → xyz/abc is a grammar then simply replace A with B's rules Hence A → xyz/abc is the result.

If there is circular reference; then generate dependency graph to find out the derivation

Eg: ~~S → A/b~~



Remove NULL productions and then UNIT prod

***

$S \to AB/\epsilon$ $A \to CD$ $B \to DC$ $D \to d$

$C \to cC/\epsilon$?

Here $C \to \epsilon$ $D \to \epsilon$ $A \to \epsilon$ ($\because A \to CD$)

$B \to \epsilon$

Taking all possible combinations where A, B, could go to ϵ

$S \to AB/B/A$

$A \to CD/C/D$

$B \to DC/D/C$

$D \to dD/d$

$C \to cC/c$

Removing unit productions

Non unit productions

$S \to AB$ $A \to CD$ $B \to CD$ $D \to dD/d$ $C \to cC/c$

dependency relations..

$S \to A$ $S \to B$ $A \to C$ $A \to D$ $S \to C$ S

$B \to C$ $B \to D$

So final grammar: $S \to AB/CD/DC/cC/c/dD/d$

$A \to CD/cC/c/dD/d$

$B \to DC/dD/d/cC/c$

$C \to cC/c$

$D \to dD/d$

Eliminate useless productions, ε-productions and unit productions from the grammar?

S → a | aA | B | C A → aB | ε B → Aa

C → bCD D → ddd

CHOMSKY - NORMAL FORM

A context free grammar $G = \langle V, T, P, S \rangle$ is said to be CNF if the productions are of the forms, either

(i) $A \to BC$ or (ii) $A \to a$

where A, B, C are non terminals and 'a' is a terminal. In CNF, no. of symbols on the right side of the productions strictly limited. Every non-empty CFL without the empty string (ϵ) can be transformed into a CNF grammar.

Theorem: Every CFL L with $\epsilon \notin L$ can be generated by a CNF grammar.

Proof: We know that if 'L' is a CFL with $\epsilon \notin L$ then there is a CFG $G = \langle V, T, P, S \rangle$ that generates 'L'. Without loss of generality we may assume that G has no ϵ production and no unit productions. First construct a grammar $G' = \langle V, T, P', S \rangle$ from G such that productions in P' are of the form

$A \to X_1 X_2 \ldots X_K$ or

$A \to a$

where each X_i, $i = 1, 2 \ldots k$ is a symbol in V and a in T.

To construct such a grammar, follow the following rules

Rule 1: If $A \to a$ is in P, then retain it in P'

Rule 2: If $A \to aB$ (or $A \to Ba$) in P then replace 'a' with a new non-terminal X_a and add a production of the form $X_a \to a$ in P'

Rule 3: If $A \to a_1 a_2 \ldots a_n B$ is in P with $n \geq 2$ replace each a_i with new non-terminals X_{a_i} where $i = 1, 2 \ldots n$ adding productions

$A \to X_{a_1}, X_{a_2} \ldots X_{a_n} B$ and

$X_{a_i} \to a_i$ for $i = 1, 2 \ldots n$ in P'

In next step, we break all productions in P' of the form $A \to X_1 X_2 \ldots X_n$ for $n > 2$ into a group of productions with two non-terminals in each body by applying the following rule. Rule 4: Introduce $(n-2)$ new non-terminals $Y_1, Y_2 \ldots Y_{n-2}$. The original production $A \to X_1 X_2 \ldots X_n$ is replaced by the $(n-1)$ new productions given as

$A \to X_1 Y_1 \quad Y_1 \to X_2 Y_2 \quad Y_2 \to X_3 Y_3 \ldots \ldots$

$Y_{n-3} \to X_{n-2} Y_{n-2}, \quad Y_{n-2} \to X_{n-1} X_{n-1}$

Q1. Find equivalent CNF

$S \to aAD$ $\quad A \to aB / bAB$

$B \to b$ $\quad D \to d$

$S \to aAD$ is not in required form

$S \to C_a D_1$ $\quad C_a \to a$

$D_1 \to AD$

$A \to aB$: $A \to C_a B$

$A \to bAB$ $\quad A \to C_b D_2$ $\quad C_b \to b$, $D_2 \to A$

$B \to b$ & $D \to d$ are in required form.

So resultant CNF

$G_2 = (\{S, A, B, D, C_a, C_b, D_1, D_2\}, \{a, b, d\}, P', S)$

where P' consists of

$S \to C_a D_1$ $\quad C_a \to a$ $\quad D_1 \to AD$ $\quad A \to C_a B$

$C_b \to b$ $\quad D_2 \to AB$ $\quad B \to b$ $\quad D \to d$.

Q2. Find equivalent CNF

(i) $S \to abAB$ $\quad A \to bAB / \in$ $\quad B \to Baa / A / \in$

Eliminating $\in$-productions

Since $A \to \in$, $B \to \in$

$S \to abAB / abA / abB / ab$

$A \to bAB / bB / bA / b$

$B \to Baa / aa / A$

Eliminating unit productions from B we get

$B \to Baa \mid aa \mid bAB \mid bB \mid bA \mid b$

→ Introducing new ~~productions~~ non terminals S_a, S_b we obtain

$S \to S_a S_b AB \mid S_a S_b A \mid S_a S_b B \mid S_a S_b$

$S_a \to a \quad S_b \to b$

$A \to S_b AB \mid S_b B \mid S_b A \mid \cancel{S_b} b$

$B \to B S_a S_a \mid S_a S_a \mid S_b AB \mid S_b A \mid S_b B \mid b.$

→ Break further productions so that the RHS contains exactly two non terminals.

Thus introducing new non terminals S_c, S_d, S_e

$S \to S_a S_c B \mid S_a S_c \mid S_a S_d \mid S_a S_b$

$S_c \to S_b A \quad S_d \to S_b B. \quad S_e \to S_a B$

$S_a \to a \quad S_b \to b$

$A \to \cancel{S_b A} S_c B \mid S_b B \mid S_b A \mid b$

$B \to S_e S_a \mid S_a S_a \mid S_c B \mid S_b A \mid S_b B \mid b$

Still the production contains more than 2 - NT's introduce a new NT S_f.

∴ the CNF is : $S \to S_a S_f \mid S_a S_d \mid S_a S_c \mid S_a S_b$

$S_c \to S_b A \quad S_d \to S_b B \quad S_e \to BS_a \quad S_f \to S_c B.$

$S_a \to a \quad S_b \to b.$

$A \to S_b A \mid S_b B \mid S_c B \mid b$

$B \to S_e S_a \mid S_a S_a \mid S_c B \mid S_b A \mid S_b B \mid b$ //

Convert to CNF

3. $S \to ABa$ $A \to aab$ $B \to Ac$

Introduce S_a, S_b & S_c

$\therefore S \to ABS_a$ $A \to S_a S_a S_b$ $B \to A S_c$

Introduce two NT's C & D

$C \to BS_a$ and $D \to S_a S_b$

CNF is

$S \to AC$, $A \to S_a D$ $B \to AS_c$

$C \to BS_a$ $D \to S_a S_b$ $S_a \to a$ $S_b \to b$ $S_c \to c$

4. Find equivalent CNF.

$S \to bA / aB$

$A \to bAA / aS / a$

$B \to aBB / bS / b$

Introducing $S_b \to b$ $S_a \to a$.

$S \to S_b A$, $S \to S_a B$.

$A \to S_b AA$, $A \to S_a S$ $A \to a$.

$B \to S_a BB$, $B \to S_b S$ $B \to b$.

Introducing $C \to S_b A$ $D \to S_a B$.

Final set of productions

$S \to S_b A$, $S \to S_a B$, $A \to CA$, $A \to S_a S$ $A \to a$.

$B \to DB$, $B \to S_b S$, $B \to b$, $S_b \to b$, $S_a \to a$

$C \to S_b A$, $D \to S_a B$

5. $S \to \sim S \mid [S \supset S] / p \mid q$?

- $S \to p/q$ in CNF.
- $S \to \sim S$ replaced by $S \to AS$, $A \to \sim$
- $S \to [S \supset S]$ replaced by $S \to BSCSD$
  $B \to [$, $C \to \supset$, $D \to]$
- $S \to BSCSD$ replaced By $S \to BC_1$ $C_1 \to SCSD$
  $C_1 \to SC_2$ $C_2 \to CC_3$ $C_3 \to SD$.

Final : $S \to p/q/AS/BC$
$A \to a$, $B \to [$, $C \to \supset$, $D \to]$, $C_1 \to$
$C_2 \to CC_3$ $C_3 \to SD$ //

GREIBACH NORMAL FORM

This normal form does not put only restriction on the length of body of production but also put restriction on the position in which terminals and non terminals appear in body of production.

Definition: A CFG $G = \langle V, T, P, S \rangle$ is said to be in GNF if all productions are of the form $A \to ax$ where

$a \in T$, $A \in V$ and $X \in V^*$

Features:

(i) If we use GNF grammar production into a sentential form, an I/P string of length 'n' will have derivation of exactly 'n' steps through GNF grammar.

(ii) It is used ~~to~~ construct PDA for a given CFG

Procedure for constructing GNF

Eg: Let $G = S \to aSb \mid aA \quad A \to Aa \mid Sa \mid a$

Step 1: Rename non terminals of grammar in a sequence form like $A_1, A_2 \dots A_n$.

→ Now $G' = A_1 \to aA_1b \mid aA_2 \quad A_2 \to A_2a \mid A_1a \mid a$

ie Renamed S & A as A_1 & A_2 respectively

Step 2: Construct an equivalent grammar G_1' for G in which every production body for A_i starts with a terminal or starts with a_j for some $j \geq i$. For this following lemma is applied.

Lemma 1: Let G be a CFG with rules as $A \to xBy$ with $B \to b_1 | b_2 | \ldots b_m$ then we can replace $A \to xBy$ by

$A \to xb_1y | xb_2y | \ldots xb_my$

Applying this lemma to our example:

$A_2 \to A_1 a$ should be replaced by A_1's productio

ie, $A_2 \to aA_1ba | aA_2a$

$\therefore$ we obtain G_1 as.

$A_1 \to aA_1b | aA_2$ $A_2 \to A_2a | a$ & $A_2 \to aA_1ba | aA_2a$

Step 3: Apply following lemma to the grammar we got from step 1 & 2.

Lemma 2: Let A_i productions in G_1 are

$A_i \to A_i x_1 | \ldots | A_i x_m$ with

$A_i \to b_1 | b_2 \ldots | b_p$

we can introduce a variable B_i and replace for a group of m; A_i productions by

$A_i \to b_1 B_i | b_2 B_i \ldots | b_i B_i$ and

$B_i \to x_1 B_i | x_1 | x_2 B_i | x_2 \ldots x_m B_i | x_m$

Back to example: Now our grammar is

$A_1 \to aA_1b \mid aA_2$ $\quad A_2 \to A_2a \mid a \mid aA_1ba \mid aA_2a$

(applying lemma 2 here)

Here $A_2 \to A_2a$ and other A_2 productions

$A_2 \to a \mid aA_1ba \mid aA_2a$; by applying lemma 2 here we get

$A_2 \to aB_2 \mid aA_1baB_2 \mid aA_2aB_2$ where

$B_2 \to aB_2 \mid a$.

Step 4: Finally use lemma 1 once again starting from the productions for A_n then A_{n-1} and so on down to A_1. After all A_i productions have RHS that starts either with terminal or A_j for $j > i$, we get A_i productions in GNF. Likewise B_i productions can be fixed.

So the ~~GNF~~ of our example question

$A_1 \to aA_1b \mid aA_2$

$A_2 \to a \mid aA_1ba \mid aA_2a$

$A_2 \to aB_2 \mid aA_1baB_2 \mid aA_2aB_2$

$B_2 \to aB_2 \mid a$

Now apply the technique for the production of CNF to obtain GNF.

Convert into GNF

$A_1 \to A_2 A_3$ (1) $A_2 \to A_3 A_1 / b$ (2) $A_3 \to A_1 A_2 / a$ (3)

Here 1 & 2 satisfies $A_i \to A_j \gamma$ where $j > i$

But 3 is not satisfying above condition.

---(3) $A_3 \to A_1 A_2 / a$

Substitute $A_1 \to A_2 A_3$ in this eqn

$A_3 \to A_2 A_3 A_2 / a$ — (3) again condition not satisfy

so substitute $A_2 \to A_3 A_1 / b$ in (3); we get

$A_3 \to A_3 A_1 A_3 A_2 / b A_3 A_2 / a$.

Here recursion is present. Hence apply Lemma 2

$A_3 \to b A_3 A_2 B / a B / b A_3 A_2 / a$

$B \to A_1 A_3 A_2 B / A_1 A_3 A_2$.

Here A_3 productions are of GNF. Now apply A_3 productions on A_2

So $A_2 \to A_3 A_1 / b$ will be ... $A_2 \to b$,

$A_2 \to b A_3 A_2 A_1 / a A_1 / b A_3 A_2 B A_1 / a B A_1$

Then substitute these A_2 productions in A_1

$A_1 \to A_2 A_3$ will be

$A_1 \to b A_3 A_2 A_1 A_3 / b A_3 A_2 B A_1 A_3 / a A_1 A_3$

$/ a B A_1 A_3 / b A_3$.

Replace A_1 productions in B

$B \to b A_3 A_2 A_1 A_3 A_3 A_2 / b A_3 A_2 B A_1 A_3 A_3 A_2 / a A_1 A_3 A_3 A_2$

$B \to a B A_1 A_3 A_3 A_2 / b A_3 A_3 A_2$

$B \to b A_3 A_2 A_1 A_3 A_3 A_2 B / b A_3 A_2 B A_1 A_3 A_3 A_2 B$

$B \to a A_1 A_3 A_3 A_2 B / a B A_1 A_3 A_3 A_2 B / b A_3 A_3 A_2 B$.

Convert to GNF

3. $S \to BS$ $A \to bc$ ~~$B \to AC$~~ $S \to Aa$ $A \to b$

~~$A \to b$ & $B \to a$ are in GNF.~~

Let us replace S, A, B by A_1, A_2, A_3 respectively

$A_1 \to A_3 A_1$ ~~$A_2 \to A_3 A_1$~~ $A_3 \to A_2 C$

$A_1 \to A_2 a$ $A_2 \to bc$

$S \to ASc \quad S \to Ab \quad A \to SA \quad A \to c$

Let S & A are replaced by A_1 & A_2, the productions are

$A_1 \to A_2 A_1 c$

$A_1 \to A_2 b \quad A_2 \to A_1 A_2 \quad A_2 \to c$

From production $A_2 \to A_1 A_2$; we replace A_1 by

$A_2 \to A_2 A_1 c A_2 / A_2 b A_2$

Here Recursion apply lemma 2

$A_2 \to c B_1 / c B_2$

$B_1 \to A_1 c A_2 B_1 / A_1 c A_2$

$B_2 \to b A_2 B_2 / b A_2$

Now apply A_2 productions in A_1

$A_1 \to c B_1 A_1 c / c B_2 A_1 c$

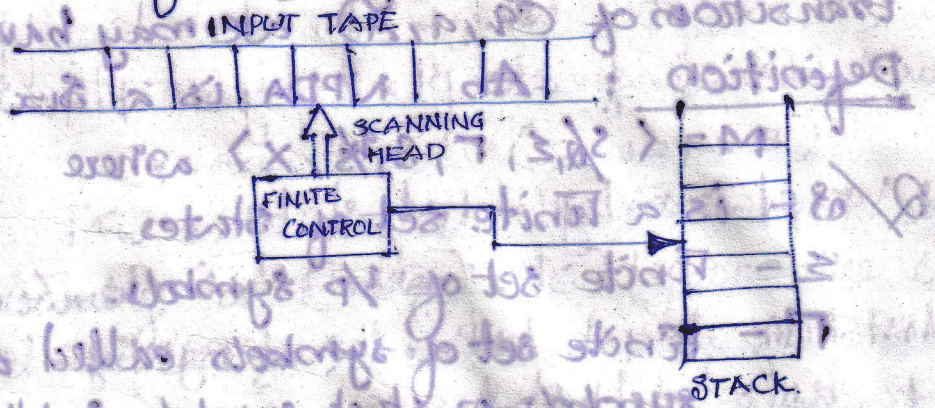
$A_1 \to c B_2 A_1$

PUSHDOWN AUTOMATA

Pushdown Automaton (PDA) is a powerful device needed for accepting arbitrary CFLs.

NPDA (Non-Deterministic Push Down Automata)

An NPDA is like finite automata that is augmented by auxiliary memory that is structured as stack. Stack can be used as a storage mechanism that can contain unbounded number of items.



The scanning head reads the i/p tape from left to right, the control unit is in one of the states and the stack contains stack symbols. Each move of NPDA is a function of scanned symbol 'a', present state 'q'

and the symbol on the top of the stack, 'A' that is, of (q, a, A) which is called ID (Instantaneous Description) of a PDA. It consists of 3 steps

1. Scanning head moves to right
2. Control unit commuting to a new state 'q'
3. Change the top of the stack with a (possibly empty) string of stack symbol.

In NPDA, we may also perform a move without scanning any symbol, that is, in the transitions of (q, a, A) we may have $a = \lambda$ (emp

Definition : An NPDA is a six tuple $M = (S, \Sigma, \Gamma, \delta, q_0, X)$ where

Q/S - is a finite set of states.

Σ - Finite set of I/P symbols.

Γ - Finite set of symbols called auxiliary symbols or stack symbols such that $\Gamma \cap \Sigma = \phi$

δ - Transition mapping defined by

$$\delta : S \times (\Sigma \cup \{\lambda\}) \times \Gamma \longrightarrow S \times \Gamma^*$$

q_0 / $S \in S$ is initial state of control unit.

Z0 / $X \in \Gamma$ is initial stack symbol.

Instantaneous description of a PDA.

In response to an i/p symbol (or ε) PDA goes from configuration to configuration. A PDA's configuration involves a state and content of stack.

Moves in PDA

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z)$ be a PDA

Define $\vdash_P$ or just $\vdash$ as follows

Suppose $\delta(q, a, X)$ contains (p, α). Then for all strings w in Σ^* and β in Γ^*

$$(q, aw, X\beta) \vdash (p, w, \alpha\beta)$$

This move reflect the idea that by consuming 'a' (may be ε) from the input and replacing X on the top of the stack by α, we can go from state q to state p.

Here w (rest of i/p) and β (rest of stack symbol) do not influence the action of PDA.

Languages of a PDA

A PDA accepts its i/p by consum
it and entering an accepting state. This approa
is called "accepting by final state."

A second approach to define a
PDA by "accepting by empty stack", that is
the set of strings that cause the PDA to emp
its stack, starting from the initial ID.

Two methods are equivallent in the
sense that language 'L' has a PDA that
that accept it by final sate iff 'L' has
a PDA that accept it by empty stack.

Acceptance by Final state

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be
a PDA then $L(P)$, the language accepted b
P by final state, is

$$\{w \mid (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \alpha)\}$$

for some state q in F, any stack string α.

Acceptance by Empty stack.

For each PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F$
we also define $N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \epsilon,$

for any state q, i.e. $N(P)$ is a set of i/ps w that P can consume and at the same time empty its stack.

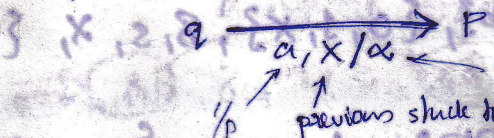
GRAPHICAL NOTATION FOR PDA

In transition diagrams of a PDA:

(a) The node correspond to the state of PDA

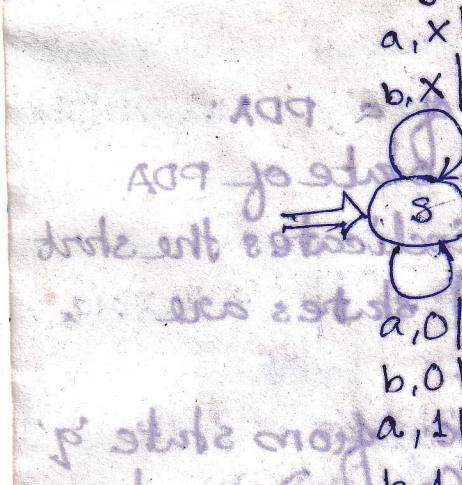
(b) An arrow labelled start indicates the start state, and doubly circled states are accepting states for PDA.

(c) An arc labelled $a, X/\alpha$ from state 'q' to state 'p' means that $\delta(q, a, X)$ contains the pair (p, α), perhaps among other pairs.



1. Construct an NPDA, M for the language $L = \{w \in \{a,b\}^* \mid n_a(w) = n_b(w)\}$ s.t. $L = L$

Sol: Insert a counter symbol '0' into st whenever an 'a' is read, then pop one counter symbol from stack when one 'b' is f



δ

$\delta(s, \lambda, X) = \{(q_f$

$\delta(s, a, X) = \{(s,$

$\delta(s, b, X) = \{(s,$

$\delta(s, a, 0) = \{(s,$

$\delta(s, b, 0) = \{(s,$

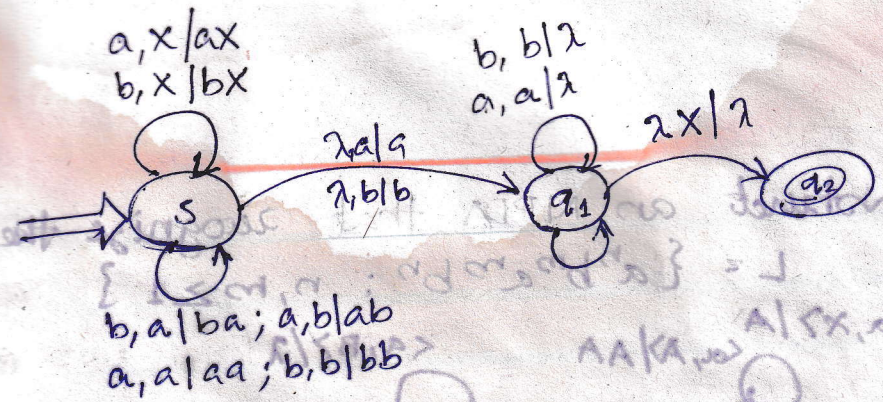
$\delta(s, a, 1) = \{(s,$

$\delta(s, b, 1) = \{(s,$

$M = \{\{s, q_f\}, \{a,b\}, \{0,1,X\}, \delta, s, X, \{q_f\}\}$.

2. Construct an NPDA for $L = \{w c w^R \mid w \in \{a$ s.t. $L = L(M)$.

Sol: When reading first part of the s we push consecutive symbols on the stac For second part, we compare the curren i/p symbol with top of the stack, contin as long as they match.



δ

$\delta(s,a,a) = \{(s,aa)\} \mid \delta(s,b,a) = \{(s,ba)\} \mid \delta(s,a,b) = \{(s,ab)\}$

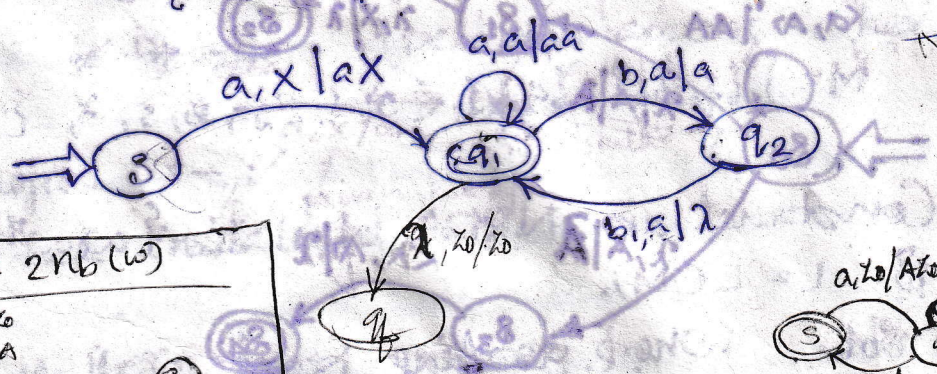
$\delta(s,b,b) = \{(s,bb)\} \mid \delta(s,a,x) = \{(s,ax)\} \mid \delta(s,b,x) = \{(s,bx)\}$

$\delta(s,\lambda,a) = \{(q_1,a)\}$

$\delta(s,\lambda,b) = \{(q_1,b)\}.$

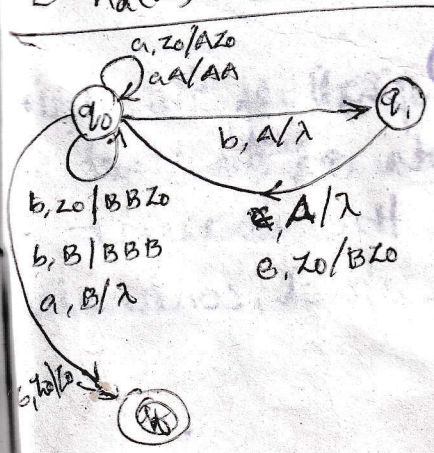
$\delta(q_1,a,a) = \{(q_1,\lambda)\} \mid \delta(q_1,b,b) = \{(q_1,\lambda)\} \mid \delta(q_1,\lambda,x) = (q_2,\lambda)$

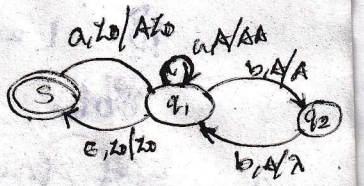
3. Construct an NPDA for language $L = \{a^n b^{2n} \mid n \geq 0\}$ s. $L = L(M)$.

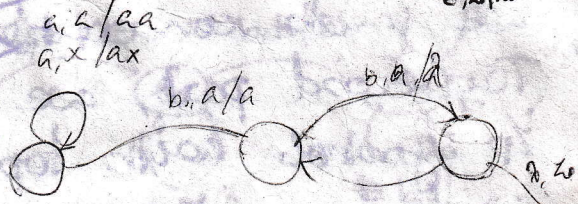


NPDA

$L = n_a(\omega) = 2n_b(\omega)$

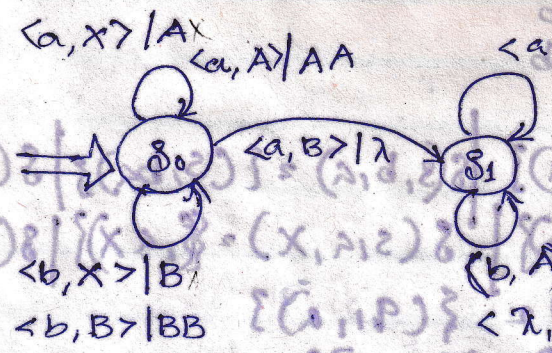




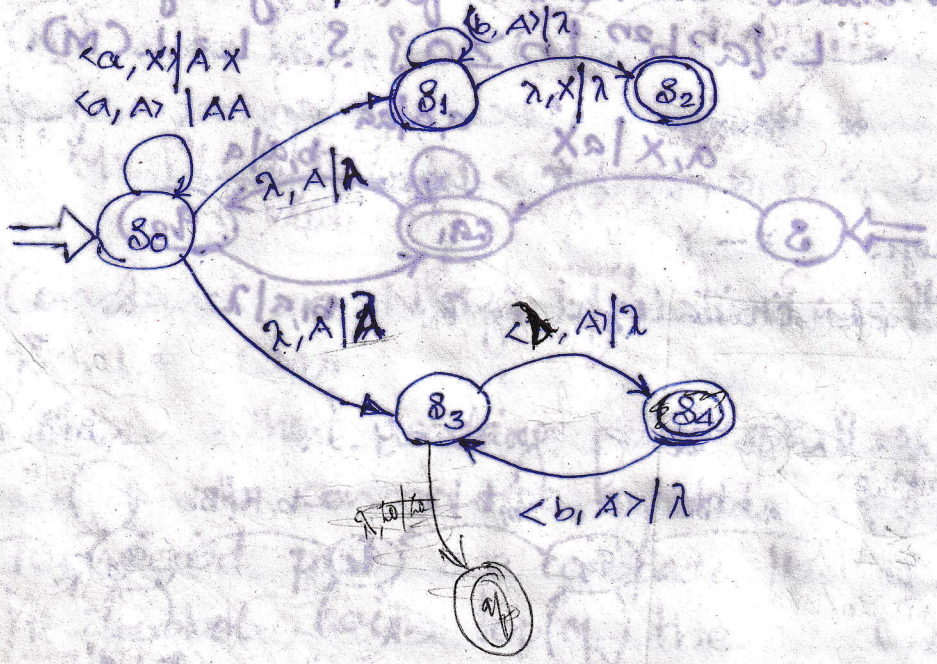


4] Construct an NPDA that recognize the language

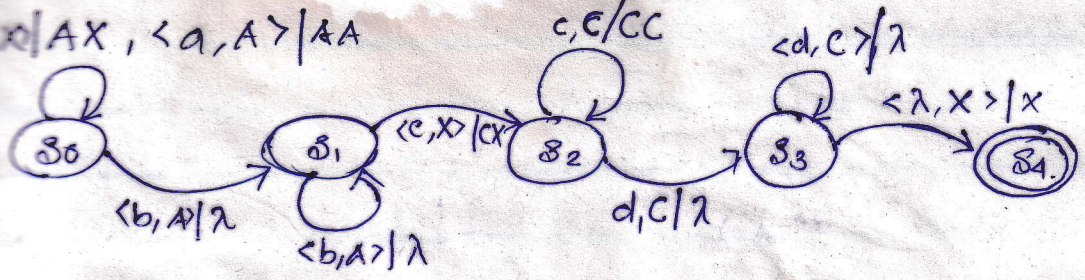
$L = \{ a^n b^m a^m b^n : n, m \geq 1 \}$



5) $L = \{ a^n b^m \mid n \geq 1 \text{ and } n = m \text{ or } n = 2m \}$

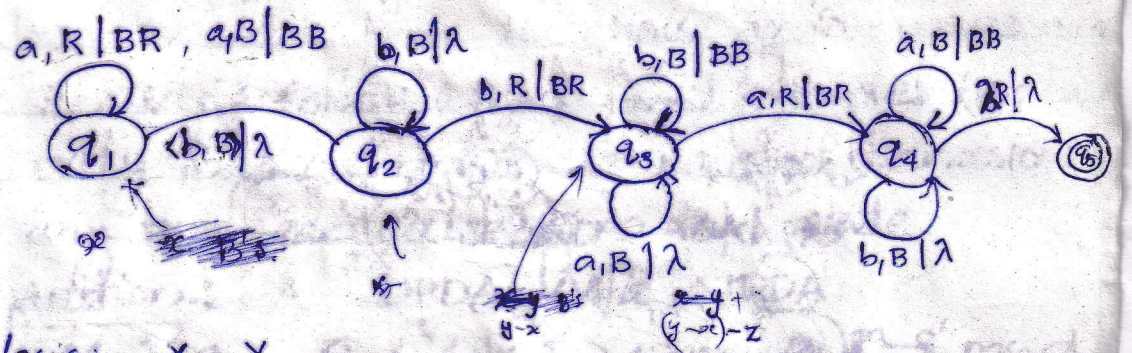


2. $L_5 = \{ a^n b^n c^m d^m : n, m \geq 1 \}$ Via Final state



y - x

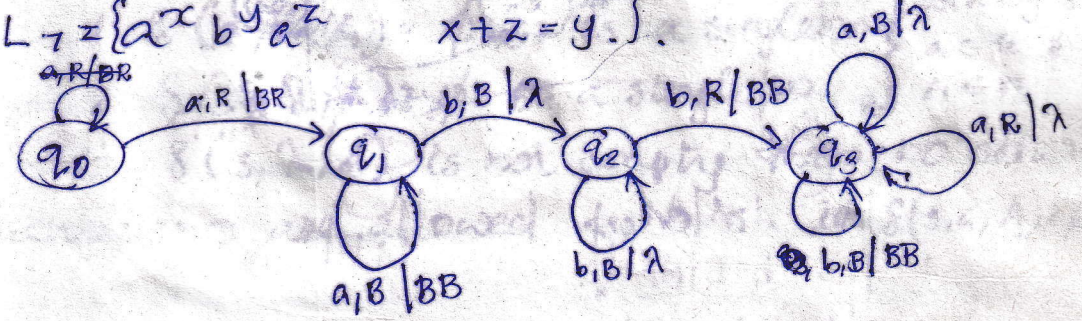
7. $L_6 = a^x b^y a^z b^w$ where $x+z = y+w$ ~~Lab stack~~



Logic: $x - y$

Stack initialized by R.

8. $L_7 = \{ a^x b^y a^z \quad x+z = y. \}$.



DETERMINISTIC PDA

DPDA adds no more computing power to NPDA. There exists languages which are accepted by NPDA but not by DPDA.

A DPDA differs from NPDA at following points.

1. For a given i/p symbol, DPDA can't have 2 or more different choices of states or two different choices of strings to push into stack.
2. DPDA can't have the choice of λ-moves either.
3. In DPDA states that have λ-moves can have only that one move, ie no other transitions of any type can exist from that state.

Definition: A DPDA is a NPDA $M = \langle S, \Sigma, \Gamma, \delta, s, X, F \rangle$ where δ must fulfill following criterions:

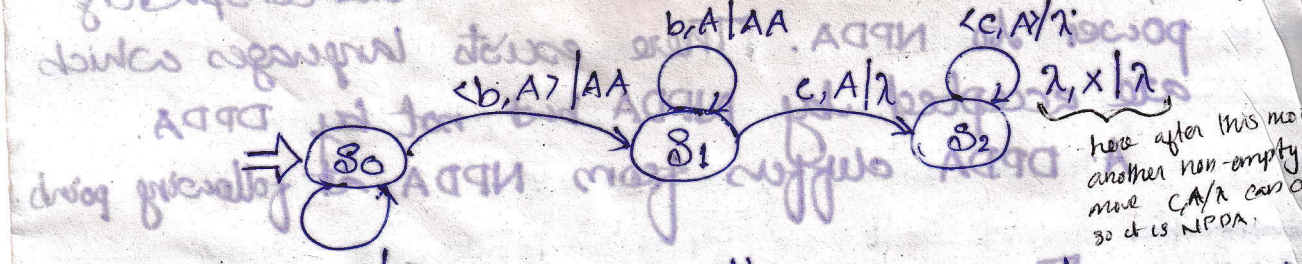
1. $\delta(s, a, A) = \phi$ or is a singleton
2. $\delta(s, \lambda, A) = \phi$ or a singleton

} For every $s \in S$, $a \in \Sigma$ & $A \in \Gamma$

3. $\delta(s, \lambda, A)$ is not empty then no other transitions are allowed from 's' ie $\delta(s, a, A) = \phi$

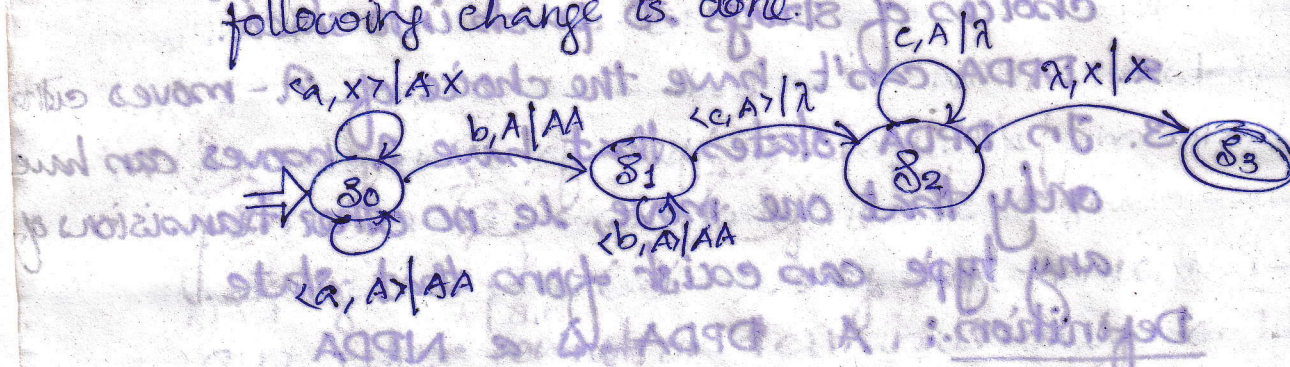
1. Let $L = \{a^i b^j c^k \mid i, j, k \geq 1 \text{ and } i+j=k\}$

Find NPDA & PDA that accepts L.

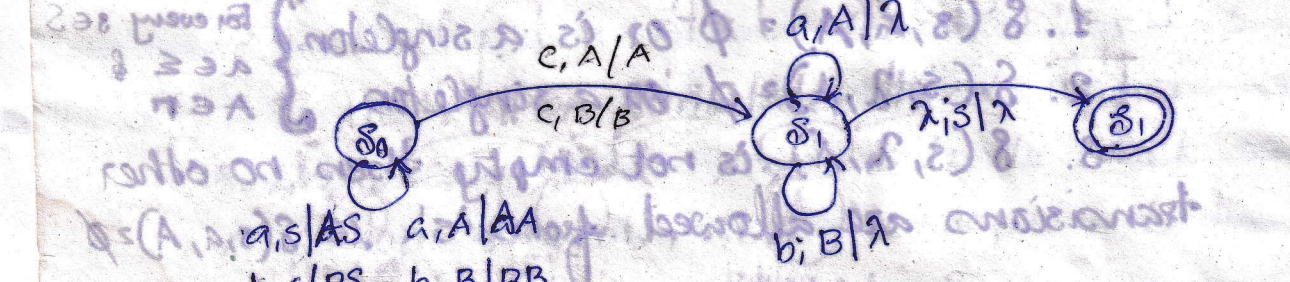


a, X|AX
a, A|AA

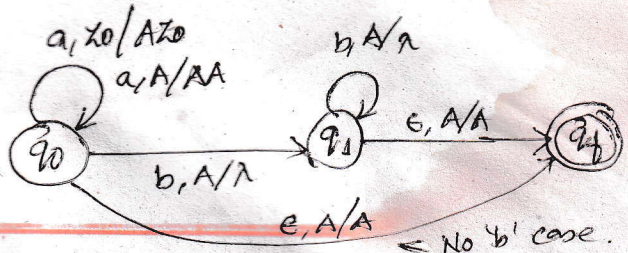
In this NPDA, rule 3 is violate
So for making it a DPDA ~~accept~~
following change is done.



2. $L = \{wcw^R \mid w \in \{a,b\}^*\}$



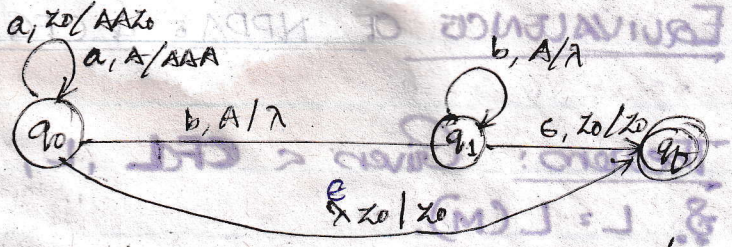
a,S|AS a,A|AA
b,S|BS b,B|BB
a,B|AB b,a|BA.
b,B|λ



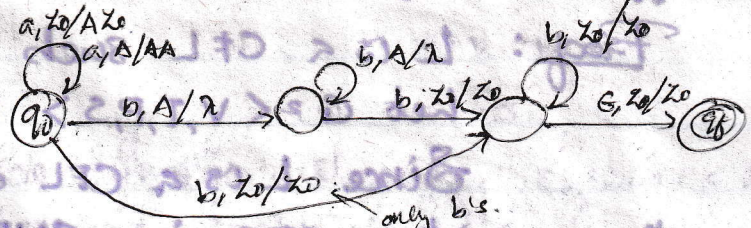
$n > m$
$m \geq 0$

← No 'b' case.

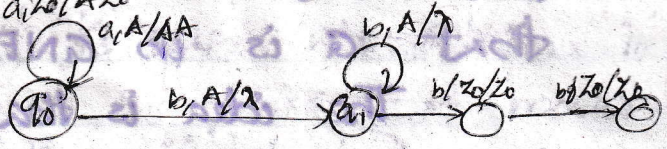
② $a^n b^{2n}$ $n \geq 0$



⑤ $a^b b^m$ $b < m$



⑥ $a^b b^m$; $m \geq b+3$



EQUIVALENCES OF NPDAs & CFGs

Theorem: Given a CFL, L, there exist a NP
s.t. $L = L(M)$.

Proof: L is a CFL such that $\lambda \notin L$

Let $G = \langle V, T, P, S \rangle$ be a CFG which gener

Since L is a CFL without NULL stri
there exists a CFG in GNF in it. So assum
that G is in GNF.

The idea is that the NPDA simulate
the sequence of left sentential forms that
grammar 'G' uses to generate a given term
string $\omega \in L(G)$. Any left sentential f
can either consists of terminals only or be
a type $xA\alpha$ where A is the leftmost
non terminal 'x' is the string of terminals
and α is the string of terminals and non termin
respectively.

→ The NPDA simulates the derivation in
such a way that $A\alpha$ is in the stack with
A as top symbol, and 'x' represents the
corresponding prefix of the I/p string.

The NPDA $M = <\{s_0, s_1, s_f\}, T, V \cup \{X\}, \delta, s_0, X, \{s_f\}>$

Where δ is defined by following rules

1. $\delta(s_0, \lambda, X) = \{s_1, sX\}$: This will move start symbol s on the top of the stack.
2. $\delta(s_1, a, A) = \{(s_1, \alpha) \mid$ if $A \to a\alpha \in P\}$ (a: first terminal on the body of production)
   ie if the i/p symbol and top of stack symbol are equal the rest of the body of the production is now on top of stack
3. If $\delta(s_1, \lambda, X) = \{s_f, X\}$: NPDA moves to final state.

1. Construct a PDA that accepts the language generated by grammar

   $S \to aSbb / a$. This is not in GNF

   So changing the production to

   $S \to aSA / a$

   $A \to bB$

   $B \to b$

Step 1: First move start symbol on to the top of stack

$\delta(q_0, \lambda, z_0) = \{(q_1, Sz_0)\}$

Step 2: The production $S \to aSA$ is simulated by replacing 'S' on the top of the stack by 'SA' while

reading i/p symbol a. 111 the rule S→ should cause the PDA to read an 'a' ch simply removing 'S'. ie

$$\delta(q_1, a, S) = \{(q_1, SA), (q_1, a)\}$$

In this manner the other productions are

$$\delta(q_1, b, A) = \{(q_1, B)\}$$

$$\delta(q_1, b, B) = \{(q_1, a)\}$$

$$\delta(q_1, \in, Z_0) = \{(q_2, \in)\}$$ //

2. Construct NPDA for

$S \to aA$, $A \to aABC / bB / a$

$B \to b$, $C \to c$

$$\delta(q_0, \in, Z_0) = \{(q_1, SZ_0)\}$$

$$\delta(q_1, a, S) = \{(q_1, A)\}$$

$$\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \in)\}$$

$$\delta(q_1, b, A) = \{(q_1, B)\}$$

$$\delta(q_1, b, B) = \{(q_1, \in)\}$$

$$\delta(q_1, c, C) = \{(q_1, \in)\}$$

$$\delta(q_1, \in, Z_0) = \delta(q_f, Z_0)\}$$

3. Construct NPDA for

$S \to aABB \mid aAA$, $A \to aBB \mid a$ $B \to bBB \mid A$.

Since there is a unit productions in B ; rewrite

$S \to aABB \mid aAA$ $A \to aBB \mid a$ $B \to bBB \mid aBB \mid a$

$\delta(s_0, \epsilon, z_0) = \{(s_1, Sz_0)\}$

$\delta(s_1, a, S) = \{(s_1, ABB), (s_1, AA)\}$.

$\delta(s_1, a, A) = \{(s_1, BB), (s_1, \epsilon)\}$

$\delta(s_1, a, B) = \{(s_1, BB), (s_1, \epsilon)\}$.

$\delta(s_1, b, B) = \{(s_1, BB)\}$

$\delta(s_1, \epsilon, z_0) = \delta(s_f, z_0)$

PUMPING LEMMA FOR CONTEXT-FREE LANGUA

Pumping lemma for CFL says in any sufficiently long string in a CFL it is possible to find atmost two short, nearby substrings, that we can pump.

Statement

Pumping lemma for CFL is similar to pumping lemma for regular languages, but we break each string 'z' in CFL 'L' into five parts and we pump the second and fourth in tandem.

Theorem: Let L be a CFL. Then there ex a constant 'n' such that if 'z' is any str in L such that |z| is at least n, then we can write z = uvwxy subject to th following condition

1. $|vwx| \leq n$ ie, middle portion is not too long.
2. $|vx| \neq \epsilon$ Since v & x are the pieces to pumped. ie at least one of the st are pump must not be empty
3. For all $i \geq 0$ uv^iwx^iy is in L. ie too

PUMPING LEMMA FOR CONTEXT-FREE LANGUA

Pumping lemma for CFL says in any sufficiently long string in a CFL it is possible to find atmost two short, nearby substrings, that we can pump.

Statement

Pumping lemma for CFL is similar to pumping lemma for regular languages, but we break each string 'z' in CFL 'L' into five parts and we pump the second and fourth in tandem.

Theorem: Let L be a CFL. Then there ex a constant 'n' such that if 'z' is any str in L such that |z| is at least n, then we can write z = uvwxy subject to th following condition

1. $|vwx| \leq n$ ie, middle portion is not too long.
2. $|vx| \neq \epsilon$ Since v & x are the pieces to pumped. ie at least one of the st are pump must not be empty
3. For all $i \geq 0$ uv^iwx^iy is in L. ie too

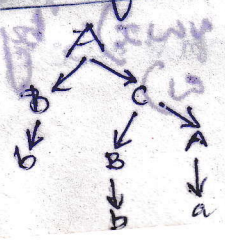
strings v & x may be pumped any no. of times including zero, and the resulting string will still be a member of L.

Proof: First step is to find a CNF of grammar G for L. If $L = \emptyset$ then the statement for the theorem, which talks about a string z in L surely can't be violated, since there is no such z in $\emptyset$. Also the CNF grammar G will generate $L - \{\epsilon\}$ and we pick $n > 0$, in which case 'z' can't be ϵ anyway.

Let $G = \langle V, T, P, S \rangle$ such that $L(G) = L - \{\epsilon\}$ and G have m variables. Choose $n = 2^m$. Suppose 'z' is in L is of length at least 'n'. Then the parse tree whose longest path is of length 'm' or less must have yield of length 2^{m-1} (ie $n/2$) or less. Such a parse tree cannot have yield 'z' as 'z' is too long (of length $\geq n$). Thus any parse tree with yield 'z' has a path of length at least $m+1$. $\Rightarrow$

Ex: $A \to BC/a$
$B \to BA/b$
$C \to BA.$

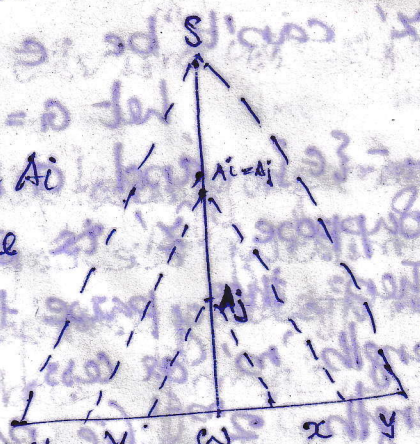
$A \overset{*}{\Rightarrow} w = bbca$



path length = 3 $\quad m = 3$

The longest path in tree for z (where yield will be of length atleast 2^m) should have path of length k+1 where $k \geq m$. In the path there are atleast m+1 occurances of variables $A_0, A_1 \ldots A_k$. But there are 'm' different variables in V. That means atleast two of last m+1 variables on the path must be same variables. Suppose $A_j = A_i$ where $k-m \leq i < j \leq k$.

Then we can divide the tree as shown.



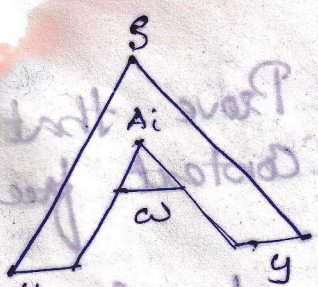
String 'w' is the yield of subtree rooted at A_j

Yield of subtree rooted at A_i is vwx. Since there are no unit productions, v & x could not both be 'ε', although one could be.

If $A_i = A_j = A$.

First replace the subtree rooted at (which has yield vwx) by subtree rooted at (which has yield w)

resulting tree :-

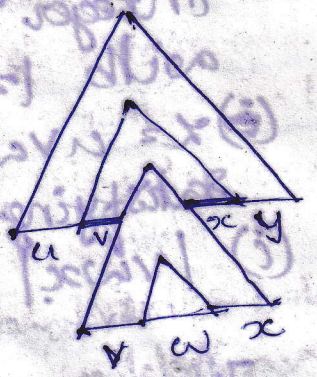


$= uwy$

$= uv^0wx^0y$

Another option is that subtree rooted at Aj by Ai

the resulting tree is :-



yield is uv^2wx^2y

i.e uv^iwx^iy where $i \geq 2$

Similarly we could have a tree with yield uv^3wx^3y and so on

1 Prove that the following language is no
Context free language.

$$L = \{ a^n b^n c^j / n \leq j \leq 2n \}$$

Suppose L is context-free.
Then by pumping lemma, there is a t
integer 'n' and that for any $z \in L$
with $|z| \geq n$ can be decomposed as
(i) $z = uvwxy$ (ii) where $u\ v\ w\ x\ y\ z \in \{$
satisfying following
(i) $|vwx| \leq n$ (ii) $|vx| \geq 1$ (iii) and $uv^i wx^i y \in L$

If we decompose 'z' satisfying conditio
(i) & (ii) respectively
then vwx consists of only one symbol o
straddles of two symbols from a's, b's or c
respectively.
→ If vwx consists of b's only, then
uwy consists of n a's and 2n c's b
fewer than 'n' of b's. Thus uwy cannot b
→ If vwx straddles two symbols say
a's and b's then we can choose 'i' such

$uv^i wx^i y$ has more than $2n$ occurances of (a or b) and exactly $2n$ occurances of c.
Thus $uv^i wx^i y$ cannot be 'L', for some 'i'.
∴ L is not CFL.

2. $L = \{a^P \mid P \text{ is a prime}\}$. Is it a CFL?
Suppose L is CFL.
Let P is a prime number such that $P > n$.
Let us decompose 'z' into $z = uvwxy$ satisfying condition 1 & 2 of P.L.
By property 3 of PL, $uv^i wx^i y$ is in CFL for all $i = 0, 1, 2 \ldots$
Hence $uwy \in L$ and $|uwy| = m$ is a prime no.
consider $|uv^m wx^m y| = m + m|vx| = q$.
i.e $q = m(1 + |vx|)$ since $|vx| \geq 1$;
q is a multiple of m and q is not prime.
Since $uv^m wx^m y$ is not in CFL; L is not a CFL.

3. $L = \{0^i 1^j 2^i 3^j \mid i \geq 1 \text{ and } j \geq 1\}$
Let $z = 0^n 1^n 2^n 3^n$
Follow the steps of question no. 1.

4. $L = \{ww \mid w \text{ is in } \{0,1\}^*\}$

Let $w = 0^n 1^n$ then $z = 0^n 1^n 0^n 1^n$

we can break z such that $|vwx| \leq n$ $vx \neq \epsilon$. We shall show that uwy not in L.

Since $|vwx| \leq n$, $|uwy| \geq 3n$.

Thus if uwy is some repeating string tt then..

1. → Suppose vwx is within first n 0's ...

Let vx consist of k 0's where $k > 0$

then ~~first~~ uwy begins with 0^{n-k} 1'

Since $|uwy| = 4n-k$ we know that if $uwy = tt$ then $|t| = 2n - k/2$. ie t ends but uwy ends in 1. so it can't equal tt

2. → Suppose vwx ~~consists of~~ straddles the ~~end~~ block of 0's and first block of 1's. It may be that vx consists only 0's if Then the argument that uwy is not of tt is the ~~as~~ same as case 1. If vx at least one 1, then we note that t must end in 1^n becoz uwy ends in 1^n However there is no block of n ~~ones~~ 1's ex the final block.

⇒ If vwx contained in first block of 1's, then the argument that uwy is not in L' is like the second part of case 2.

⇐ Suppose vwx straddles the first block of 1's and second block of 0's. If vx actually has no 0's then the argument is the same as if vwx were contained in first block of 1's. If vx has at least one 0, then uwy start with a block of n 0's and so does t if uwy = tt. However there is no other block of n 0's in uwy for second copy of t.

⇒ In other cases, where uwy in the second half of z', the argument is symetric to the cases where vwx is contained in first half of z.

Closure properties of CFL

CFLs are closed under Union, Concatenation & Kleen star opn

Union

If L_1 and L_2 be two CFLs then L_1 is also context free.

Let $L_1 = \{a^n b^n, n>0\}$ Corresponding gramm G_1 will have $P_1: S_1 \to aS_1b/ab$

Let $L_2 = \{c^m d^m, m \geq 0\}$ in G_2, $P_2: S_2 \to cS_2d/$

Union of L_1 & L_2, $L = L_1 \cup L_2 = \{a^n b^n\} \cup \{c^m$

Corresponding grammar G will have produc

$S \to S_1 / S_2$

Note: L_1 says no. of a's should be equal to no. of b's and L_2 says no. of c's should be equal to no. of c's. Then union says either two condition to be true.

Concatenation.

If L_1 & L_2 are CFL then $L_1 L_2$ is also CFL

ref. previous example $L = L_1 L_2$

$= \{a^n b^n c^m d^m\}$ corresponding gramm

G will have additional production $S \to S$

The concatenation (L) says first number of a's should be equal to number of b's and then number of c's should be equal to number of d's. A PDA can be designed which pushes for a's & pops for b's and then pushes c's & then pop d's.

Kleene closure

If L_1 is CFL then L_1^* is also CFL

$L_1 = \{a^n b^n \mid n >= 0\}$ then $L_1^* = \{a^n b^n \mid n \geq 0\}^*$ is also CFL.

say $S \to aSb/\epsilon$ then L_1^* will have production $S_1 \to SS_1/\epsilon$

CFL is not closed under intersection & complementation

Ex $L_1 = \{a^n b^n c^m \mid n \geq 0, m \geq 0\}$ and

$L_2 = \{a^m b^n c^n \mid n \geq 0, m \geq 0\}$

then $L_3 = L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ is not CFL.

L_1 says no. of a's should be equal to no. of b's, and L_2 says no. of b's should be equal to no. of c's, their intersection says both these conditions needed to be true ie no. of a's = no. of b's = no. of c's. It can't be accepted by a PDA. hence not CFL.

Similarly complementation of CFL L_1 is $\Sigma^* - L_1$ need not be CFL

Ex. $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ If CFL closed under complementation

$\overline{L_1}$ is CFL, $\overline{L_2}$ is CFL then $L = \overline{L_1} \cup \overline{L_2}$ is CFL.

but $L_1 \cap L_2 = \overline{L}$, ie we know that $L_1 \cap L_2$ is not CFL then $\overline{L}$ is not CFL.

Intersection with regular language

If L_1 is CFL & L_2 is regular then $L_1 \cap L_2$ is CFL.

Let $L_1 = \{a^n b^n : n \geq 0\}$.

$L_2 = \{a^{10} b^{10}\}$.

$L_1 \cap L_2 = a^{10} b^{10}$ is regular.

$= \{a^n b^n : n = 10\}$.

TURING MACHINES.

The limitation of finite state machine is that they do not have arbitrary large memory i.e no capacity of remembering a large amount of information.

Push down automata has an external memory capable of remembering long sequence of I/p's, but it fails to accept all languages due to its strict Last In First Out policy.

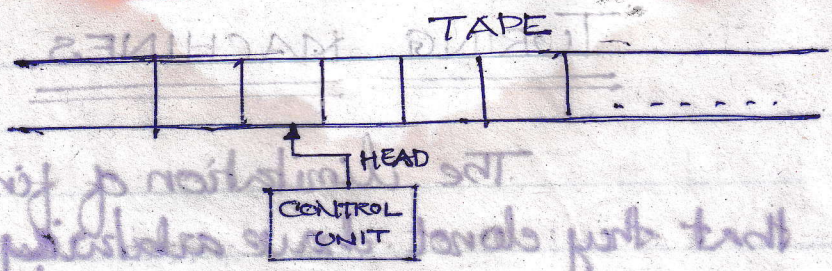
Turing machine is a refinement of finite state machine which retains the finite state control but provides in addition the facility to read as well as write data on the single tape of infinite length.

THE ~~TURING~~ STANDARD TURING MACHINE.

Turing machine consists of two main components.

→ An infinite tape in both directions

→ The control unit.



Tape is a sequence of cells that contai
symbols from a finite alphabet.
The control unit can perform ~~unit~~ move
depending on i/p symbols under its hea
and its internal state:
A move by a control unit consis
→ changing the internal state
→ Rewriting the symbol under
→ Moving tape head left(L) or
or keeping it stationary (N)

Formal definition

A standard Turing machine T is given
$T = \langle S, A, \delta, S_0, F \rangle$
S :- Finite set of internal states including HA
A :- Finite tape alphabets including a speci
symbol 'B' known as Blank sym
S_0 :- Starting state. F :- set of final

- Transition mapping defined by

$(S-F) \times A \longrightarrow S \times A \times \{R, L, N\}$

CONFIGURATION OR INSTANTANEOUS DESCRIPTION OF TM

Action of a TM can be described by a set of quintuples of the form $\langle s, a, b, s', d \rangle$ where 's' is the present state, 'a' is the tape symbol being read, b is the symbol written in place of 'a', s' is the next state and d denote the direction in which R/W head moves.

Non-deterministic T.M.

A Turing machine has atleast two configurations with the same s and a symbol then TM is non-deterministic. otherwise it is deterministic.

If Turing machine gets into a configuration for which its present state and symbols read are not in the first two symbols of any quintuple the T.M halts.

Language accepted by T.M

A TM said to be accept an I/P string if it enters the halt state otherwise

T.M rejects the input string, In the case there are two possibilities - Ei T.M stops by entering a state other H Halt state or it runs forever.

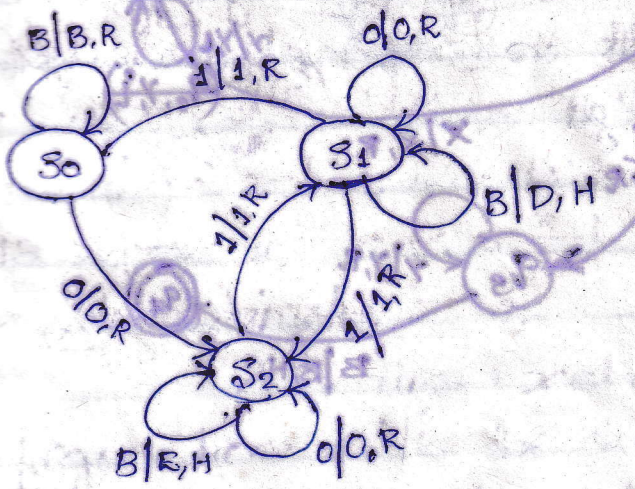
Definition.

A T.M $T_A = \langle S, A, \delta, S_0, F \rangle$ recognize or accept a language $L (\subseteq A$ if it halts in a final state for all input strings x in L., i.e for all x in L:

So $x \vdash^{*}_{T_A} \alpha S_f \beta$ for some $S_f \in F$ an any string $\alpha, \beta \in A$

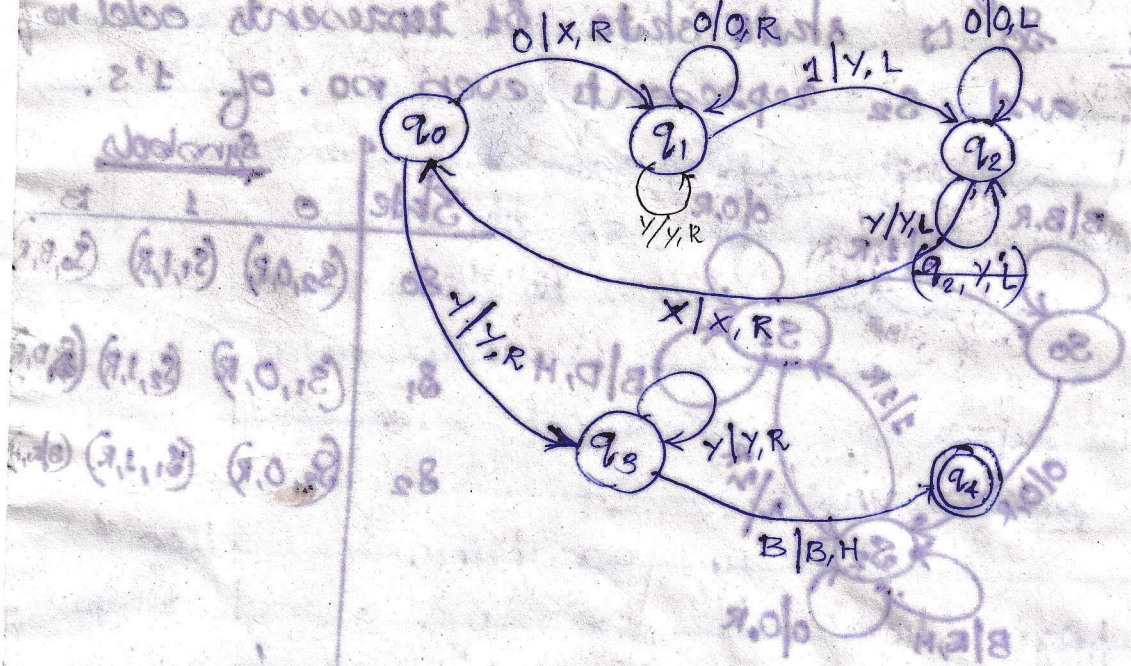
1. Design a T.M that examines a specified string of 0's and 1's on a tape and print 'E' when the no. of 1's are even and 'D' if the no. of 1's are odd.

Sol: S_0 is start state S_1 represents odd no. of 1's and S_2 represents even no. of 1's.

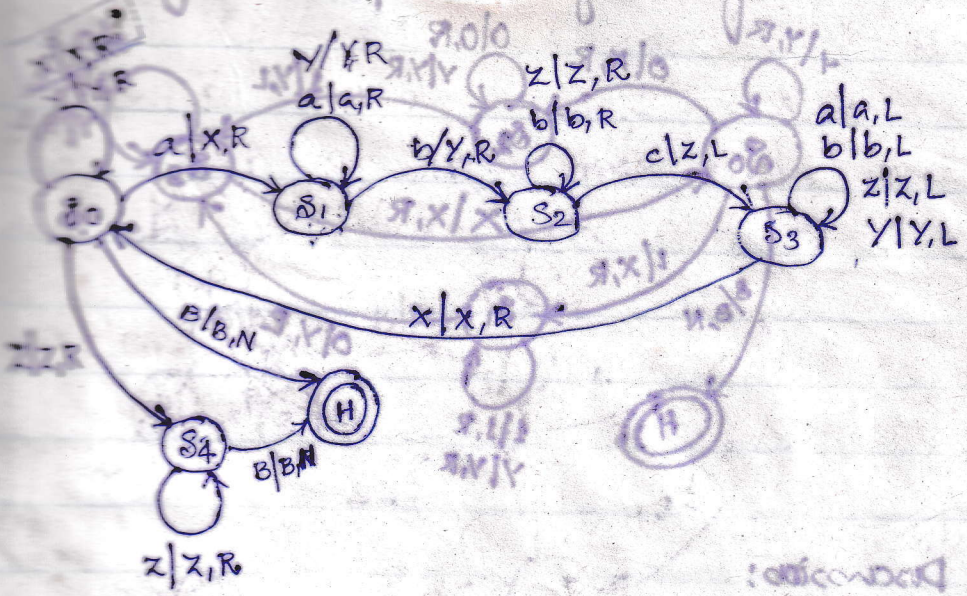


State	Symbols 0	1	B
S_0	$(S_2, 0, R)$	$(S_1, 1, R)$	(S_0, B, R)
S_1	$(S_1, 0, R)$	$(S_2, 1, R)$	(S_1, D, R)
S_2	$(S_2, 0, R)$	$(S_1, 1, R)$	(B, E, H)

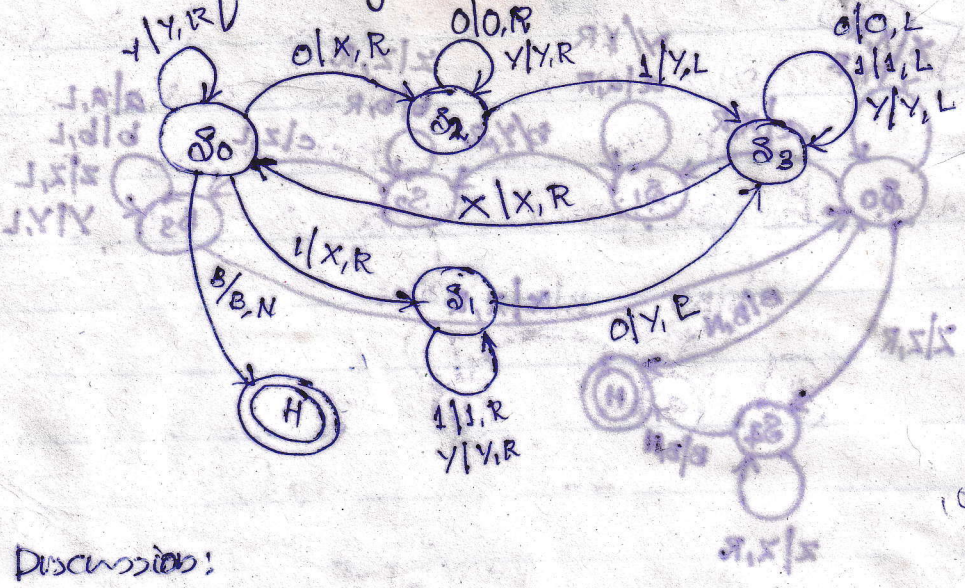
2. Design a T.M which accepts the la
$\{0^n 1^n / n \geq 1\}$



Construct a TM that recognizes the language $\{a^n b^n c^n \mid n \geq 0\}$



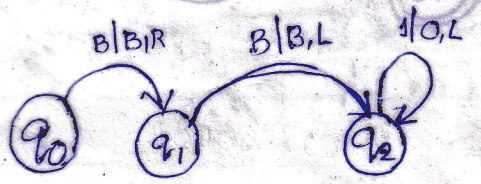
4. Design a TM for the language "set of strings with equal no. of 0's & 1's"



Discussion:

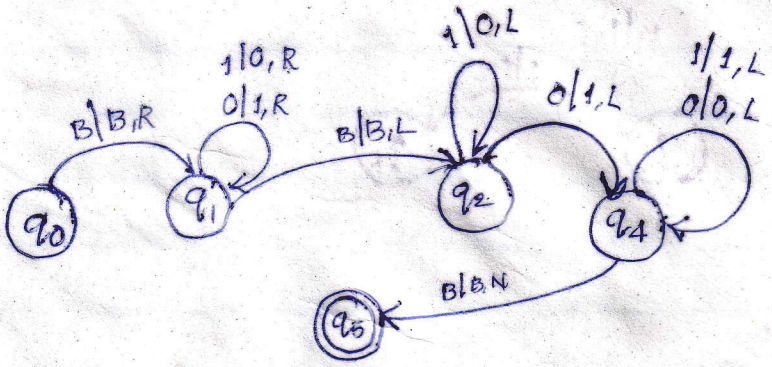
Turing machine that finds two's complement of given number:

Sol: Move towards right till the first B. At the right end, start moving towards left till you get first one and then move towards left and change each '1' by 0 and 0 by 1



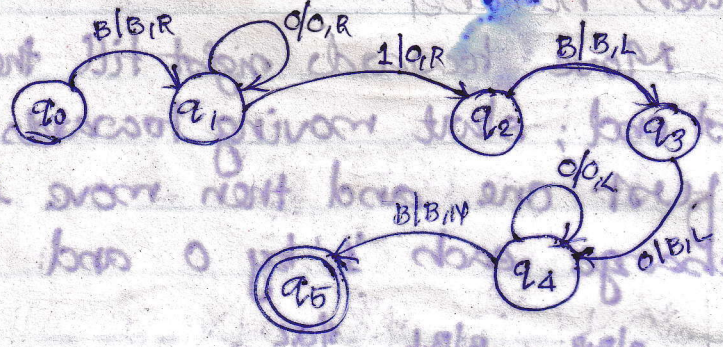
OR

8



Another method shown in figure: On the movements to right Each bit is complemented. At the ~~end~~ right end, it returns back converting one's to zroes and on finding first '0', it stops ~~com~~ the process of making 1's to zroes. The remains left part will be correct values.

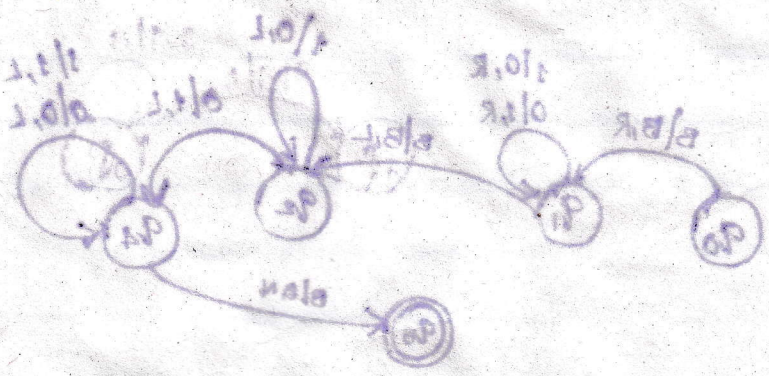
6. Turing machine that adds two binary nu



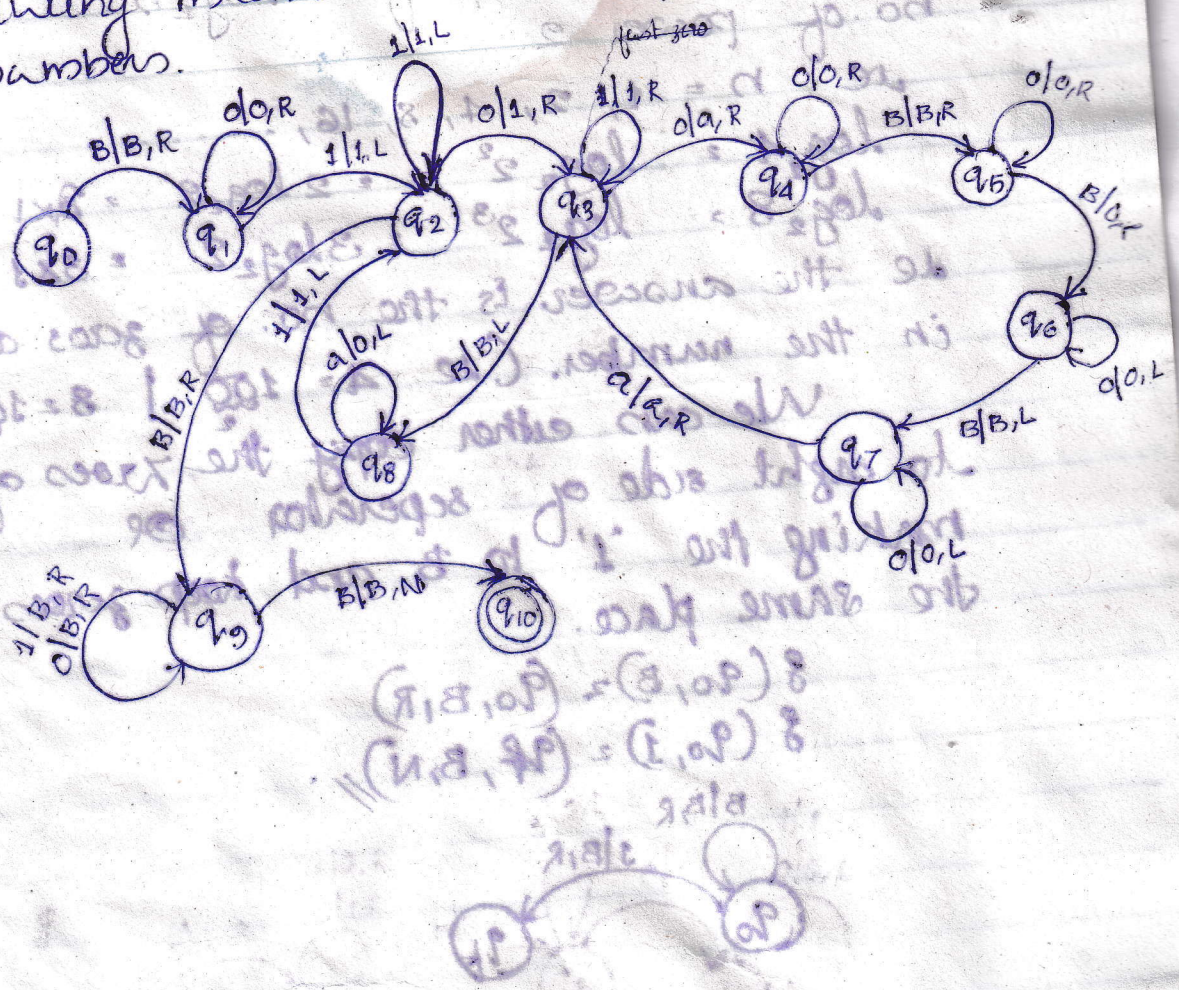
No 4+5

B-B 0000 100000BB... ⟹ 0000 00000B

4 separator 5



7. Turing machine that multiplies two ~~binary~~ unary numbers.



8. Design a T.M to perform $\log_2 n$. where 'n' is no. of power 2.

ie $n = 1, 2, 4, 8, 16, \ldots$

$\log_2 4 = \log_2 2^2 = 2\log_2 2 = 2 \times 1 = 2$

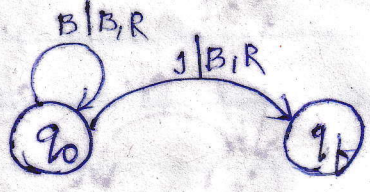
$\log_2 8 = \log_2 2^3 = 3\log_2 2 = 3 \times 1 = 3$

ie the answer is the no. of zeros after in the number. (ie $4 = 100$ | $8 = 1000$)

We can either copy the zeroes after to right side of seperator or making the '1' to B and keep zeroes the same place.

$\delta(q_0, B) = (q_0, B, R)$

$\delta(q_0, 1) = (q_f, B, N)$ //



MULTIPLE TRACK TURING MACHINE

Is a multiple track T.

a single tape

Important remaining portions.

Variants of Turing machine.

- Multitape, multitrack, No

*Universal TM, Multidimensional, Multih

Equivalence of Single tape & Multitap

↓

Recursive Enumerable Languages.

↓

Undecidability *

↓

Halting problem of TM

Recursive languages

A language is recursive of

a Turing machine that accepts every string of

and rejects every string over the same alpha

is not in the language. If a language

recursive; then its complement 'L' is also re

Decidability

Decidable : There is a TM which accepts or reject, No l

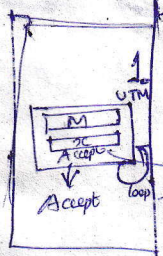
Semidecidable : There is a which can accept/reject/loop

Undecidable : There is no T Turing Machine for that language

Is there a TM co accepts the langua

M#x. (M some cha x ∀ that accep

ie.



$L_{MP} = \{M\#x \mid M \text{ accepts } x$

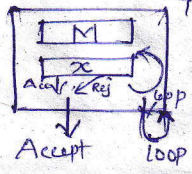
☹ ← Membership pblm

then L_{MP} is R.E langu

halting: Means either TM accepts or rejects. ★Not loop ☺

2) $L_{HP} = \{M\#x \mid M \text{ halts}$

☺ ← halting problem.



HP = { M#x | M halts on x } Is it decidable.

(ie we can design a TM for HP (ie semi decidable); but can we design a TTM for HP)

Proof by contradiction

1. Assume that there is a Total TM for HP.

   Such a TTM will say Yes when < M accepts x / M rejects x

   " " " No when < M loops on x / M not a valid TM.

Myhill - Nerode Theorem.

M-N theorem is used to prove whether or not a language L is regular and it is also used to minimization of states in DFA.

Indistinguishability: Given a language L and x, y are the strings over Σ^*, if for every string $z \in \Sigma^*$ $xz, yz \in L$ or $xz, yz \notin L$ and x, y are said to be indistinguishable over language 'L'. It is denoted by $x \equiv_L y$. $\equiv_L$ is an equivalence relation because

reflexive: for all string x, $xz \in L$ iff $xz \in L$ $\therefore x \equiv_L x$

Symmetric: Suppose $x \equiv_L y$. This means either $xz, yz \in L$ or $xz, yz \notin L$ for all $z \in \Sigma^*$. Equivalently this means $yz, xz \in L$ or $yz, xz \notin L$ for all $z \in \Sigma^*$ which implies $y \equiv_L x$

Transitive: Suppose $x \equiv_L y$ and $y \equiv_L w$. Suppose for sake of contradiction that x and w are not indistinguishable. This means there must exist some z such that exactly one of xz ~~or yz~~ and wz is member of L. Assume xz is member of L and wz is not a member of L. $xz \in L$ implies $yz \notin L$. $wz \notin L$ implies $yz \notin L$ this is a contradiction since yz can't be a member and not member of L. Therefore $x \equiv_L y$ and $y \equiv_L w \Rightarrow x \equiv_L w$.

Since $\equiv_L$ is an equivalence relation over Σ^* $\equiv_L$ partitions Σ^* into disjoint sets called equivalence classes

Myhill - Nerode Theorem: A language is regular if $\equiv_L$ partitions Σ^* into finitely many equivalence classes. If $\equiv_L$ partitions 'n' equivalence classes then minimal DFA recognizing L has exactly 'n' states.

* If x & y are indistinguishable then, they belong to same equivalence class
* If x & y belong to different equivalence classes the are distinguishable w.r.t. L.

Eg: atleast two a's class example..

class 1: {ε, b, bb, bbb. . . . }.

class 2: {ε, ba, bbaab, bbbabb. . . }.

class 3: {baa, aabb, bbbaab. . . }.

Here indistinguishable relation divided the Σ^* into three equivalence classes.

Let $x = ba$, $y = bbab$ adding $z = bb$ leads $xz = babb \notin L$ and $yz = bbabbb \notin L$.

Here x & y are indistinguishable and they lead to the same equivalence class (class 2).

Th: Prove that Hint: If we find infinitely many 'z's (for all $x \neq y$) such that xz and yz goes to diff. states then the language is not regular. and they are in diff sets; ie x & y cant go to same states. Infinite states mean, it is not regular. ie 'z' is a distinguishing extention'

Eg: $L = \{0^n 1^n \mid n \geq 0\}$ P.T it is not regular

Take set of z's ie $\{\epsilon, 0, 00, 000. . . . \} = 0^*$
ie infinitely many z's

Let $x = 0^i$ and $y = 0^j$ such that $i \neq j$

Let $z = 1^i$ ← distinguishing extention.

$xz = 0^i 1^i \in L$ } ie 1^i distinguishes 0^i & 0^j
$yz = 0^j 1^i \notin L$ } $z = 1^i$

ie infinitely many distinct equivalence classes and hence not regular.

Syllabus

CST 301 Formal Languages and Automata Theory

Module - 1 (Introduction to Formal Language Theory and Regular Languages)

Introduction to formal language theory– Alphabets, Strings, Concatenation of strings, Languages.

Regular Languages - Deterministic Finite State Automata (DFA) (Proof of correctness of construction not required), Nondeterministic Finite State Automata (NFA), Equivalence of DFA and NFA, Regular Grammar (RG), Equivalence of RGs and DFA.

Module - 2 (More on Regular Languages)

Regular Expression (RE), Equivalence of REs and DFA, Homomorphisms, Necessary conditions for regular languages, Closure Properties of Regular Languages, DFA state minimization (No proof required).

Module - 3 (Myhill-Nerode Relations and Context Free Grammars)

Myhill-Nerode Relations (MNR)- MNR for regular languages, Myhill-Nerode Theorem (MNT) (No proof required), Applications of MNT.

Context Free Grammar (CFG)- CFG representation of Context Free Languages (proof of correctness is required), derivation trees and ambiguity, Normal forms for CFGs.

Module - 4 (More on Context-Free Languages)

Nondeterministic Pushdown Automata (PDA), Deterministic Pushdown Automata (DPDA), Equivalence of PDAs and CFGs (Proof not required), Pumping Lemma for Context-Free Languages (Proof not required), Closure Properties of Context Free Languages.

Module - 5 (Context Sensitive Languages, Turing Machines)

Context Sensitive Languages - Context Sensitive Grammar (CSG), Linear Bounded Automata.

Turing Machines - Standard Turing Machine, Robustness of Turing Machine, Universal Turing Machine, Halting Problem, Recursive and Recursively Enumerable Languages.

Chomsky classification of formal languages.

Text Book

1. Dexter C. Kozen, Automata and Computability, Springer (1999)

Mark Distribution

Total Marks	CIE Marks	ESE Marks	ESE Duration
150	50	100	3 hours

Continuous Internal Evaluation Pattern:

Attendance : **10 marks**
Continuous Assessment - Test : **25 marks**
Continuous Assessment - Assignment : **15 marks**

Internal Examination Pattern:

Each of the two internal examinations has to be conducted out of 50 marks. The first series test shall be preferably conducted after completing the first half of the syllabus and the second series test shall be preferably conducted after completing the remaining part of the syllabus. There will be two parts: Part A and Part B. Part A contains 5 questions (preferably, 2 questions each from the completed modules and 1 question from the partly completed module), having 3 marks for each question adding up to 15 marks for part A. Students should answer all questions from Part A. Part B contains 7 questions (preferably, 3 questions each from the completed modules and 1 question from the partly completed module), each with 7 marks. Out of the 7 questions, a student should answer any 5.

End Semester Examination Pattern:

There will be two parts; Part A and Part B. Part A contains 10 questions with 2 questions from each module, having 3 marks for each question. Students should answer all questions. Part B contains 2 questions from each module of which a student should answer any one. Each question can have maximum 2 sub-divisions and carries 14 marks.