VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

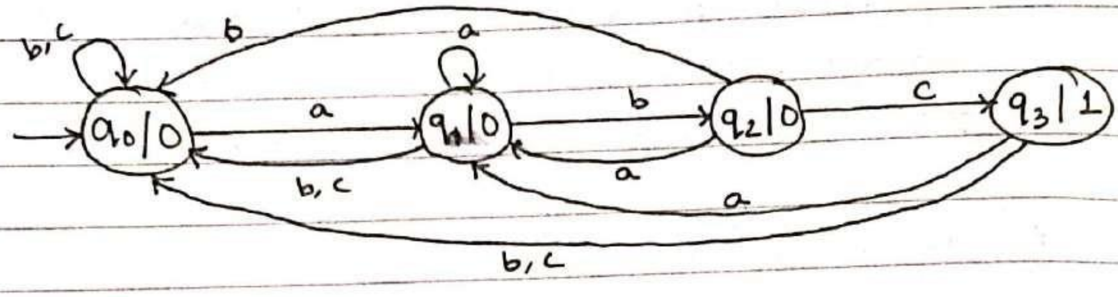
Digital Assignment, June 2020

B.Tech, Winter-2019-2020

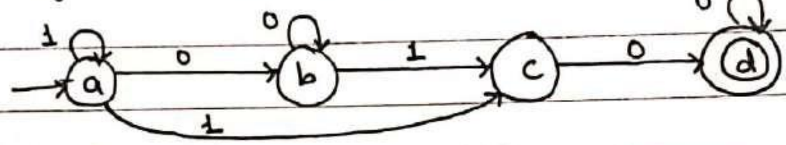
NAME	PRIYAL BHARDWAJ
REG. NO.	18BIT0272
COURSE CODE	ITE1006
COURSE NAME	THEORY OF COMPUTATION
SLOT	C1+TC1
FACULTY	Prof. HARSHITA PATEL

Q.1 A. Construct a Moore machine that takes a string consisting of a's, b's and c's as input and outputs a string containing 1 at the end of each substring abc and a 0 in all other positions. eg. i/p aabcb produces o/p 000010

A.1.A



Q.1.B Find the regular expression for the following finite state Automation



A-1-B

a : $\varepsilon + a1 + c1$ → (i)

b : $a0 + b0$ → (ii)

c : $b1$ → (iii)

d : $c0 + d0 + d1$ → (iv)

By Arden's Theorem on (ii)

b : $a00^*$ ($R = Q + RP \Rightarrow R = QP^*$) → (v)

Put (iii) in (i) ; (v) in (i) & Apply Arden's Theorem

a : $\varepsilon + a(1 + 00^*11)$ i.e a : $(1 + 00^*11)^*$

c : $(1 + 00^*11)^* 00^*1$

Apply Arden's Theorem on (iv)

d : $c0(1+0)^*$

$\therefore$ d : $(1 + 00^*11)^* 00^*10(1+0)^*$

Ans Required RE : $(1 + 00^*11)^* 00^*10(1+0)^*$

Q.2 Describe the drawback of finite automata without output. Design an NFA with no more than 5 states for the set $\{abab^n : n \geq 0\} \cup \{aba^n : n \geq 0\}$

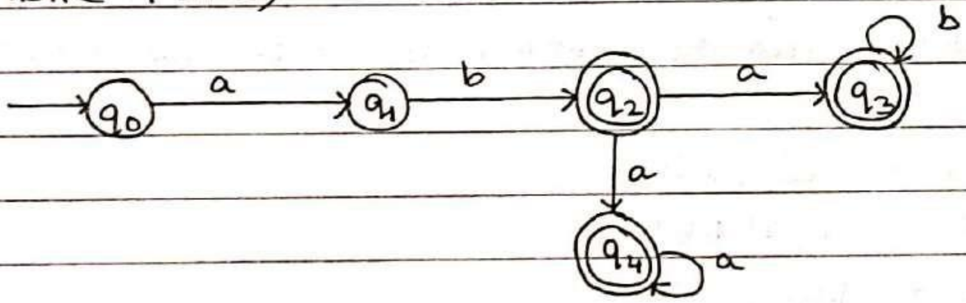
Drawbacks :-

(1) Not suited to all problem domains, should only be used when a systems behavior can be decomposed into separate states with well defined conditions for state transitions. This means that all states, transitions & conditions need to be known up-front & be well-defined.

(2) The conditions for state transitions are rigid, meaning they are fixed (this can be overcome by using a fuzzy state machine (FUSM).

(3) Larger systems implemented using a FSM can be difficult to manage & maintain without a well thought out design. The state transition can cause a fair degree of "*spaghetti factor*" when trying to follow the line of execution.

(4) The predictable nature of deterministic FSM's can be unwanted in some domain such as computer games. (solution may be non deterministic FSM's)



Q.3.A Design a finite Automata & write the regular language for the given regular expression.

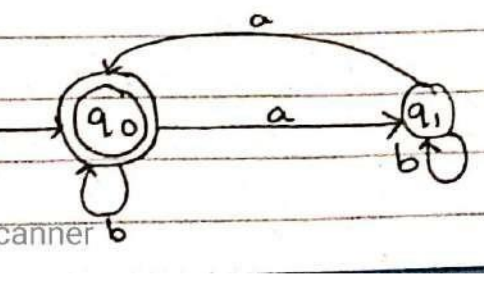
(i) $(b+(ab^*ab^*))^*$ (ii) $(0+1(01^*0)^*1)^*$

A-3.A(i) Given RE is for even number of a's or any number of b's

L_1 = even number of a's

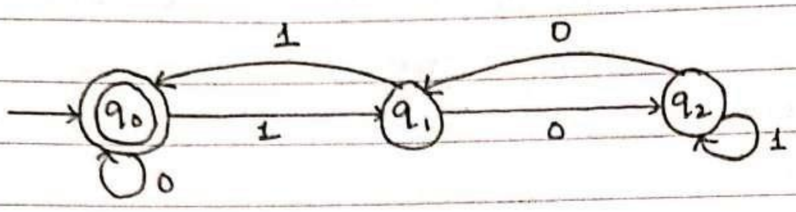
$L_2 = \{ b^n ;\ n \geq 0 \}$

RL is $L_1 \cup L_2$



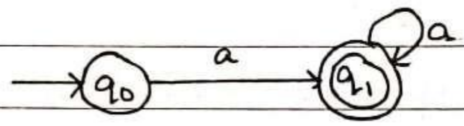
A-3-A-(ii) Given RE is for Binary Numbers divisible by 3.

eg 0, 011, 110, 1100 etc.



Q.3.B Is language $L = \{a^n \mid n \geq 1\}$ a regular language. Justify your answer.

A-3-B Given language : $L = \{a^n \mid n \geq 1\}$ is regular since a finite automata exists for it.



Q.4.A For the string aabbabab, find ~~finite~~ Leftmost derivation & rightmost derivation. Check the grammar is ambiguous or not.

A-4-A $S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bA$

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

Leftmost derivation

$S \rightarrow aB$ (by $B \rightarrow aBB$)

$S \rightarrow aaBB$ (by $B \rightarrow b$)

$S \rightarrow aabB$ (by $B \rightarrow bS$)

$S \rightarrow aabbS$ (by $S \rightarrow aB$)

$S \rightarrow aabbaB$ (by $B \rightarrow bS$)

$S \rightarrow aabbabS$ (by $S \rightarrow aB$)

$S \rightarrow aabbabaB$ (by $B \rightarrow b$)

$S \rightarrow aabbabab$

Rightmost derivation

$S \to aB$ (by $B \to aBB$)

$S \to aaBB$ (by $B \to b$)

$S \to aaBb$ (by $B \to bS$)

$S \to aabSb$ (by $S \to bA$)

$S \to aabbAb$ (by $A \to aS$)

$S \to aabbaSb$ (by $S \to bA$)

$S \to aabbabAb$ (by $A \to a$)

$S \to aabbabab$

Leftmost derivation

$S \to aB$ (by $B \to aBB$)

$S \to aaBB$ (by $B \to bS$)

$S \to aabSB$ (by $S \to bA$)

$S \to aabbAB$ (by $A \to aS$)

$S \to aabbaSB$ (by $S \to bA$)

$S \to aabbabAB$ (by $A \to a$)

$S \to aabbabaB$ (by $B \to b$)

$S \to aabbabab$

Since there is more than 1 leftmost derivation, given grammar is ambiguous.

Q.4B Convert right linear grammar to its equivalence left linear grammar.

$S \to bB$

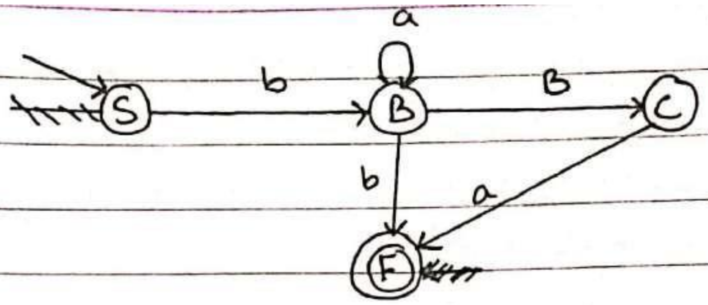
$B \to bC$

$B \to aB$

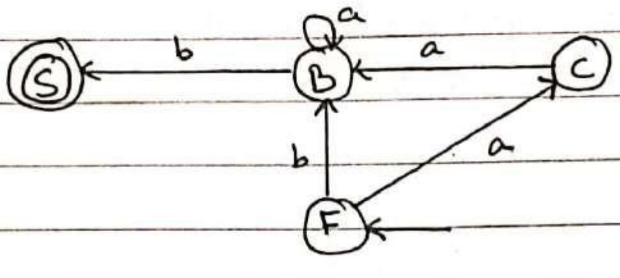
$B \to b$

$C \to a$

A-4B Step - 1: Draw finite automato for given RLG



Step 2: Interchange initial & final states & the transitions



Step 3: Write LLG from FA

$F \rightarrow Bb \mid Ca$

$C \rightarrow Bb$

$B \rightarrow Ba \mid Sb$

$S \rightarrow \varepsilon$

Q.5 Simplify the following grammar & convert it into Chomsky Normal Form.

$S \rightarrow abAB$

$A \rightarrow bAB \mid \varepsilon$

$B \rightarrow BAa \mid A \mid \varepsilon$

A.5 (1) Remove null productions

Removing $A \rightarrow \varepsilon$:

$S \rightarrow abAB \mid abB$

$A \rightarrow bAB \mid bB$

$B \rightarrow BAa \mid A \mid Ba \mid \varepsilon$

Removing $B \rightarrow \varepsilon$:

$S \rightarrow abAB \mid abB \mid abA \mid ab$

$A \rightarrow bAB \mid bB \mid bA \mid b$

$B \rightarrow BAa \mid A \mid Ba \mid Aa \mid a$

(2) Removing unit productions

Removing $B \rightarrow A$: $S \rightarrow abAB \mid abB \mid aba \mid ab$

$A \rightarrow bAB \mid bB \mid bA \mid b$

$B \rightarrow BAa \mid Ba \mid Aa \mid a \mid bAB \mid bB \mid bA \mid b$

(3) Convert grammar into CNF

Introduce new variables S_a for each $a \in T$:

$S \rightarrow S_aS_bAB \mid S_aS_bB \mid S_aS_bS_a \mid S_aS_b$

$A \rightarrow \cancel{S_bAB} \mid S_bAB \mid BS_a \mid AS_a \mid S_bB \mid S_bA \mid a \mid b \mid BAS_a$

$S_a \rightarrow a$

$S_b \rightarrow b$

(4) Introduce new variables ~~for~~ to get the first 2 productions into normal form

$S \rightarrow S_aU \mid S_aX \mid S_aY \mid S_aS_b$

$A \rightarrow S_bV \mid S_bB \mid S_bA \mid S_b$

$B \rightarrow BZ \mid S_bV \mid S_bB \mid S_bA \mid S_b \mid BS_a \mid AS_a \mid S_a$

$U \rightarrow S_bV$, $V \rightarrow AB$, $X \rightarrow S_bB$, $Y \rightarrow S_bA$, $Z \rightarrow AS_a$

$S_a \rightarrow a$, $S_b \rightarrow b$

Q.6 Explain the uses of normal forms in CFG & find the Grienbach Normal Form grammar equivalent to the following CFG.

$S \rightarrow CA \mid BB$

$A \rightarrow a$

$B \rightarrow b \mid SB$

$C \rightarrow b$

A-6 Uses:-

(1) Enables parsing: While PDA's can be used to parse words with any grammar, this is ~~given~~ often inconvenience. Normal forms can give us more structure to work with, resulting in easier parsing algorithms.

(2) Simplicity of proofs: There are plenty of proofs around CFG, including reducability & equivalence of automata. Those are the simpler, the more restricted set of grammars you have to deal with.

Therefore normal forms can be helpful there.

$S \to CA \mid BB$ Replace S with A_1

$B \to b \mid SB$ C with A_2

$C \to b$ A with A_3

$A \to a$ B with A_4

$\Rightarrow$ $A_1 \to A_2A_3 \mid A_4A_4$

$A_4 \to b \mid A_1A_4$

$A_2 \to b$

$A_3 \to a$

Now alter the rules so that the non-terminals are in ascending order.
If the production is of the form $A_i \to A_j x$ then $i<j$ & should never be $i \geq j$

$A_4 \to b \mid A_1A_4$

$A_4 \to b \mid A_2A_3A_4 \mid A_4A_4A_4$

$A_4 \to b \mid bA_3A_4 \mid A_4A_4A_4$

↳ Left ~~reductive~~ recursion

We need to remove left recursion to convert it to equivalent GNF.

Introduce new variable:

$A_4 \to b \mid bA_3A_4 \mid A_4A_4A_4$

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

$Z \to A_4A_4Z \mid A_4A_4$

Now grammar is:

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

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

$Z \to bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid bA_4Z \mid bA_3A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4$

$A_2 \to b$, $A_3 \to a$

Now the given grammar is converted to its equivalence Grienbach Normal Form.

Q.7 Define the acceptance of a PDA by empty stack. Is it true tha[t] the language accepted by PDA by empty stack or the final state [are] different languages? Design a PDA to accept the language $L = \{a^n b^m c^m d^n \mid n, m \geq 1\}$. Check the acceptance string by both empty stack & final state method.

A.7 Acceptance by Empty Stack:

On reading the input string from the initial configuration for some PDA, the stack of PDA gets empty. Let $P = (Q, \Sigma, F, S, q_0, Z, \varepsilon)$ be a PDA. The language acceptable by empty stack can be defined as :

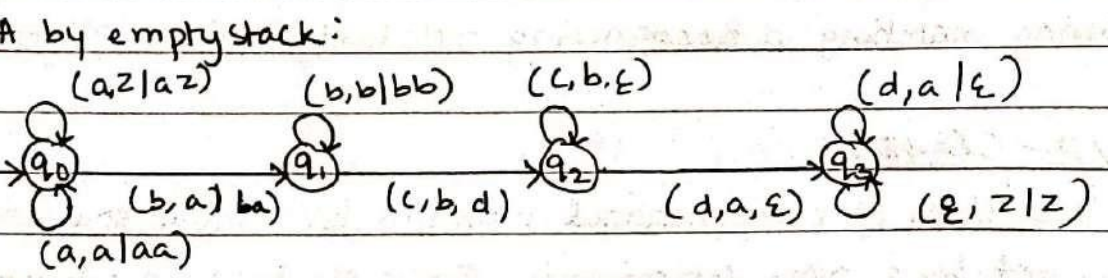
$N(PDA) = \{ \omega \mid (q_0, \omega, Z) \vdash^* (p, \varepsilon, \varepsilon), q \in Q \}$

Equivalence Acceptance by Final State & Empty Stack:

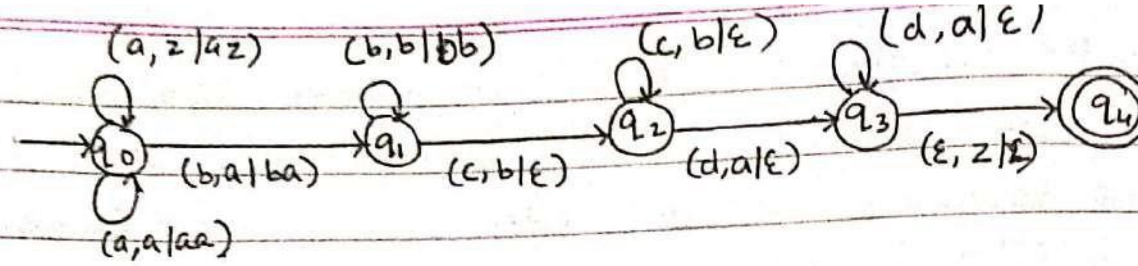
- If $L = N(P1)$ for some PDA P1, then there is a PDA P2 such that $L = L(P2)$. This means the language accepted by empty stack PDA will also be accepted by final state PDA.
- If there is a language $L = L(P1)$ for some PDA P1 then there is a PDA P2 such that $L = N(P2)$. This means language accepted by final state PDA is also accepted by empty stack PDA.

It is true that the languages accepted by PDA by empty stack or by the final state is different languages. The two don't lead to the same languages. For your language, acceptance by empty stack lead to an empty ~~language~~ language, because the stack can never be empty.

PDA by empty stack:



PDA by final state:



States are q_0, q_1, q_2, q_3, q_4.
Z means empty stack
ε means push nothing to stack
$q_4 \rightarrow$ final state

Q.8 Explain class P, NP problems & NP hard problems with suitable example.

A.8 P-Class:

→ Consists of those problems that are solvable in polynomial time, that is these problems can be solved in time $O(n^k)$ in worst case, where k is constant. These problems are call tractable while others are called intractable or super polynomial. Formally an algorithm is polynomial time algorithm if there exists a polynomial p(n) such that the algorithm can solve any instance of size n in a time $O(p(n))$

Examples Recognizing palindromes: The problem of recognizing palindromes is solvable in linear time, which is certainly polynomial time. Palindrome is string that is equal to its own reverse. For e.g abcba is palindrome,
PALINDROME = $\{x \mid x \in \{a,b,c\}^* \text{ \& } x \text{ is palindrome}\}$.
It is easy to see that PALINDROME is in P. To decide if x is a palindrome first reverse x & check whether they are equal.

other examples String matching ; Recognizing relatively prime integers.

NP-Class:

→ It is a class of computational problems for which solutions can be computed by a non-deterministic Turing Machine in polynomial time. by a
Or equivalently, those problems for which solutions can be checked in polynomial time by a deterministic TM. Another way to say this is that given a solution to this problem, it can be verified in polynomial time.

Example:- Sorting problem :- Given n integers to rearrange such that they are in non-decreasing order. This can be easily solved in O(nlogn). time. It's proposed solution is in O(n) which is polynomial in n.

NP- Hard:-

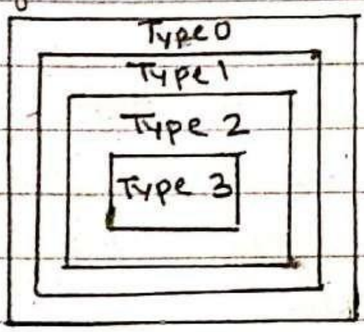
→ Problems are a class of those which are at least as hard as the hardest problems in NP. They do not have to be elements of NP, they may not even be decision problems.

Example: Given some n, find all cliques in all graphs with n vertices. Clearly this problem is harder. Note that the answer to this problem is actually all subsets of the n-vertices which form cliques. Aso note that to verify, we have a correct answer, we have to check that we have all subsets which form cliques. This problem is NP-hard but not NP- complete.

Q.9 List out the heirarchy summarized in the Chomsky heirarchy. Design a Turing Machine to compute 2's complement of any given binary number & simulate their action on the input 0100.

A-9 According to Chomsky heirarchy, grammars are divided into 4 types

Type 0	Unrestricted Grammar	- Recognized by Turing Machine
Type 1	Context sensitive Grammar	- Accepted by Linear Bound Automata
Type 2	Context Free Grammar	- Accepted by Push Down Automata
Type 3	Regular Grammar	- Accepted by Finite Automata

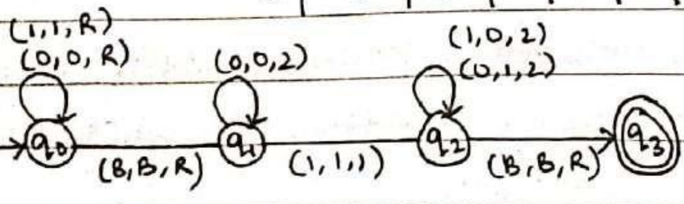


Turing Machine for 2's complements

B	B	0	1	0	0	B	B	(input)

B	B	1	1	0	0	B	B	(output)

State-Transition Diagram



Tape measurement for string "0100"

B	0	1	0	0	B

(i) Input is given as "0100" (scan using from ~~left to~~ right to left)

B	0	1	0	0	B

(ii) Pass two 0's from right

B	0	1	0	0	B

(iii) Pass 1 after that

B	1	1	0	0	B

(iv) Take complement of '0' = '1'

(v) Blank (in left) is reached when string is finished. So move to start of string (optional) by moving one-step right.

2's complement is written into TAPE in place of input string.
Input string : 0100
Output string : 1100

Q.10 State real time applications for the following notations
(i) Context Free Grammar (ii) Regular Expression (iii) DFA (iv) PDA

A-10 (i) CFG are used in compilers & in particular for parsing, taking a string-based program & figuring out what it means. Typically CFG's are used to define the high-level structure of a programming language. Figuring out how a particular string was derived tells us about its structure & meaning.

(ii) REs are useful in a wide variety of test processing tasks & more generally string processing where the data need not be textual. Common applications include data validation, data scraping, data wrangling, simple parsing, the production of syntax highlighting system.

(iii) DFA was included in protocol analysis, text parsing, video game character behaviour, security analysis, CPU control units, natural language processing & speech recognition.

(iv) PDA is used in compiler design, parser design for syntax analysis, online transaction process system & Tower of Hanoi (recursive solution)