Name: Aniruddha Shivaji Repale.

Roll No: TEIT-10

DPU

* TOC : Assignment-1 *

Que.1] Define the following terms with example.

i) Alphabet

⇒ An alphabet is a finite set of symbols or characters. These symbols are the building blocks used to construct strings and languages. Alphabet is usually denoted by Greek letter Σ (sigma).

e.g. $\Sigma = \{0, 1\}$

This alphabet contains just two symbols '0' and '1'. It is commonly used in digital systems and binary computations.

e.g. $\Sigma = \{a, b, c, \ldots\ldots, z\}$

This alphabet contains 26 symbols, representing lowercase letters of English language.

ii) String

⇒ A string is an ordered sequence of symbols chosen from a given alphabet. The length of the string is the number of symbols it contains, and it can be zero or more. An empty string, denoted by ε (epsilon), has length of zero & contains no symbols.

e.g. For the binary alphabet ($\Sigma = \{0, 1\}$)

String : ε (length 0)

String : 1 (length 1)

String : 01 (length 2)

String : 100 (length 3)

e.g. For the english alphabets ($\Sigma = \{a, b, c, \ldots\ldots, z\}$)

String : abc (length 3)

String : hello (length 5)

String : zara (length 4)

String : a (length 1)

String : ccc (length 3)

String : ε (length 0)

iii) Language

⇒ A Language in the context of formal languages and automata theory is a set of strings over a specific alphabet. A language can be finite or infinite. Languages can be described by various methods, including regular expressions, grammars & automata.

e.g. Language over the binary alphabet $\Sigma = \{0, 1\}$

$L = \{'0', '1', '01', '10', '001', '100', '1001', \ldots\}$

e.g. Language of all binary strings ending in '01'.

$L = \{'01', '101', '001', '0001', \ldots\}$

e.g. Language of all strings starting with 'a', over alphabet $\{a, b\}$.

$L = \{'a', 'aa', 'ab', 'aab', \ldots\}$

iv) DFA

⇒ A Deterministic Finite Automata (DFA) is a theoretical machine used in computer science to recognize patterns or solve decision problems by processing strings from particular language. A DFA operates on finite number of states and transitions between these states based on input symbols from alphabet. The machine begins in start state & reads an input string symbol by symbol. Depending on current state & input symbol, DFA moves to another state. If it ends in an accepting (final) state after processing the entire string, the string is said to be accepted by DFA, otherwise it is rejected.

A DFA is formally defined by following components:

Q : A finite set of states.

Σ : A finite alphabet of input symbols.

δ : A transition function. $[\delta = Q \times \Sigma]$

q_0 : Initial state

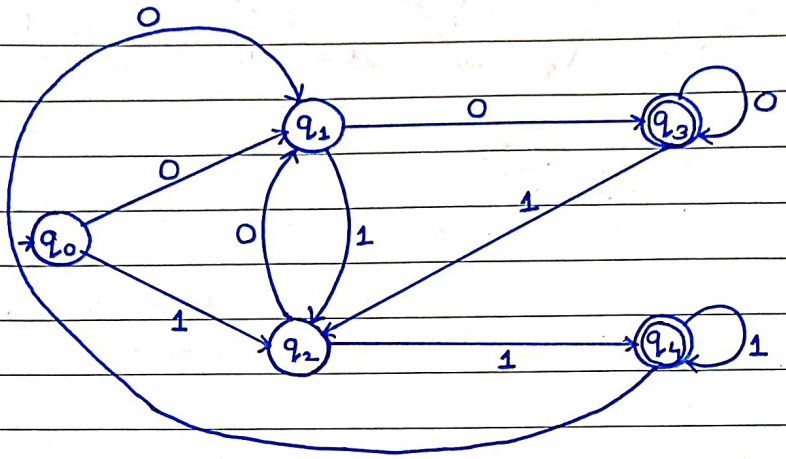
F : Final state

DPU

Que.2] Construct a DFA for the following :

i) L = {0, 1 | Accept all the strings ending in 00 or 11}

⇒ Ans :



$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

$\delta = Q \times \Sigma = Q'$

$q_0 = q_0$

$F = \{q_3, q_4\}$

Transition Table

States \ I/P	0	1
q_0	q_1	q_2
q_1	q_3	q_2
q_2	q_1	q_4
q_3	q_3	q_2
q_4	q_1	q_4

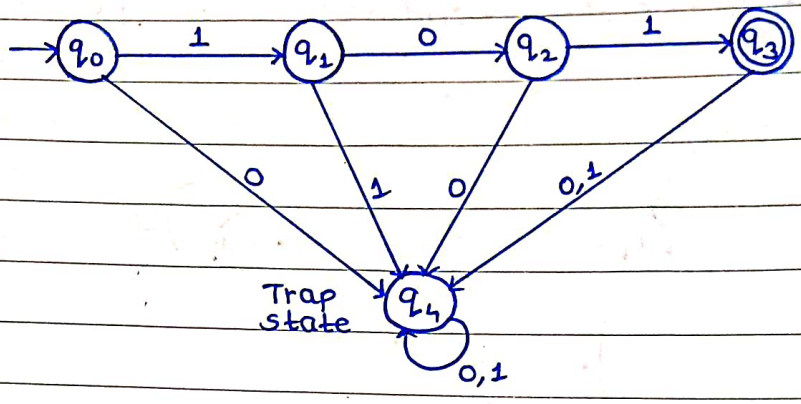
String verification (10011)

$\delta\{q_0, 10011\} = \delta\{q_2, 0011\} = \{q_1, 011\} = \{q_3, 11\} = \{q_2, 1\} = \{q_4\}$

$\because q_4 \in F$, $\therefore$ string is accepted.

ii) Design a DFA with $\Sigma = \{0,1\}$ that accepts only input 101.

⇒ Ans :



$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma = \{0,1\}$

$\delta = Q \times \Sigma = Q'$

$q_0 = q_0$

$F = q_3$

Transition Table

States \ I/P	0	1
q_0	q_4	q_1
q_1	q_2	q_4
q_2	q_4	q_3
q_3	q_4	q_4
q_4	q_4	q_4

String verification (101)

$\delta\{q_0, 101\} = q_1$

$\delta\{q_1, 01\} = q_2$

$\delta\{q_2, 1\} = q_3$

$\because q_3 \in F$, $\therefore$ string is accepted !

String verification (10011)

$\delta\{q_0, 10011\} = q_1$

$\delta\{q_1, 0011\} = q_2$

$\delta\{q_2, 011\} = q_4$

$\delta\{q_4, 11\} = q_4$

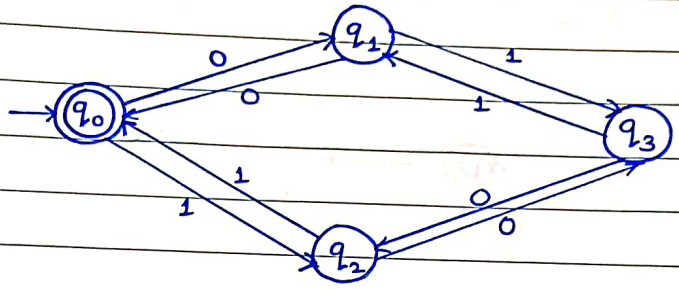
$\delta\{q_4, 1\} = q_4$

$\because q_4 \notin F$, $\therefore$ string is rejected !

DPU

iii) Design a DFA with $\Sigma = \{0, 1\}$ that accepts even number of 0's & even number of 1's.

⇒ Ans :



$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

$\delta = Q \times \Sigma = Q'$

$q_0 = q_0$

$F = q_0$

Transition Table

States \ I/P	0	1
q_0	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

String verification (1010)

$\delta\{q_0, 1010\} = q_2$

$\delta\{q_2, 010\} = q_3$

$\delta\{q_3, 10\} = q_1$

$\delta\{q_1, 0\} = q_0$

$\because q_0 \in F$, $\therefore$ string is accepted.

String verification (1011)

$\delta\{q_0, 1011\} = q_2$

$\delta\{q_2, 011\} = q_3$

$\delta\{q_3, 11\} = q_1$

$\delta\{q_1, 1\} = q_3$

$\because q_3 \notin F$, $\therefore$ string is rejected.

Name: Aniruddha

Roll No: TEIT-10

DPU

* TOC : Assignment - 2 *

Que.1] Define the following terms :

a] Kleene Closure

⇒ **Ans :**

i) The kleene closure of a set A, denoted as A^*, is the set of the all strings that can be formed by concatenating zero or more strings from A.

ii) This operation allows the repetition of elements in A any number of times, including zero repetitions (which results in empty string ε)

iii) If A is a set of strings, then

$$A^* = \{\varepsilon\} \cup A \cup A^2 \cup A^3 \cup \ldots\ldots$$

$$= \{x_1 x_2 \ldots\ldots x_n \mid n \geq 0 \text{ and } x_i \in A \ \forall\ 1 \leq i \leq n\}$$

Here, A^2 represents the strings obtained by concatenating two strings from A; A^3 represents concatenation of three strings & so on.

The empty string ε is always included in A^*.

iv) Example :

Let $A = \{a, b\}$

The kleene closure A^* consists of :

ε (empty string)

a, b (each element of a A)

aa, ab, ba, bb (all possible combinations of two elements).

aaa, aab, abb, bbb,... (all possible combinations of 3 elements) & so on.

$\therefore A^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, \ldots\ldots\}$

v) In Finite Automata (FA), A^* is used to define regular languages where any number of repetitions of certain states or transitions are allowed.

e.g. If an automata accepts 'a' then a^* means it can accept any no. of a's.

b] Positive closure :

⇒ Ans :

i) The positive closure of set A, denoted as A^+, is the set of all strings that can be formed by concatenating one or more strings from A. It is similar to kleene closure but excludes the empty string ε.

ii) This operation allows repetition of elements in 'A' any no. of times, except zero.

iii) If A is a set of strings then,

$$A^+ = A \cup A^2 \cup A^3 \cup \ldots\ldots$$

$$= \{x_1, x_2 \ldots\ldots x_n \mid n \geq 1 \text{ and } x_i \in A \text{ for all } 1 \leq i \leq n\}$$

iv) Example :

Let $A = \{a, b\}$

$\therefore A^+ = \{a, b, ab, ba, bb, aaa, aba, aab, \ldots\ldots\}$

v) In Finite Automata, A^+ defines languages where at least one repetition of certain states or transitions is required.

e.g. If an Automata accepts one 'a', then 'a+' means it must accept one or more a's, but not zero.

Que.2] Illustrate in English, the language of the following regular expression.

$$(1+\varepsilon)(00^*1)^*0^*$$

⇒ Ans :

Components of given regular expression :

i) $(1+\varepsilon)$

- It means 'either 1 or the empty string ε.'
- It indicates that string can either start with character 1 or with nothing at all.

ii) (00^*1)

- It means a single zero, followed by any no. of 0's (including no

iii) (00*1)*

It means that the entire sequence 00*1 can repeat zero or more times.

iv) 0*

It means that string can end with any no. of zero's (including none).

Language Defined:

The language generated by this regular expression consists of strings that:

- optionally start with 1.
- contains zero or more patterns of '0 followed by any number of 0s & then a 1'
- And optionally end with any no. of zeros.

Examples:

1, ε, 0, 10, 01, 001, 101, 0001, 100010, etc.

Que.3] Explain in brief applications of regular expressions.

⇒ Ans:

i) Text Searching and Pattern matching:

- Regex are widely used for searching specific patterns in text. For example, finding email addresses, phone numbers or keywords in large bodies of text.
- e.g. searching for an email addresses in a document using regex like \b[A-Z a-z 0-9 . %+-]+@[A-Z a-z 0-9 . -]+\.[A-Z|a-z]{2,}\b

ii) Input validation:

- Regex is used to validate user inputs, ensuring they follow a specific pattern. This is common in web forms where fields like email, phone no. or postal code need to meet certain criteria.

- e.g. validating an email with a regex pattern ensures the format is correct before submitting form.

iii) Data Extraction :
- Regex can extract specific data from text or logs. They are helpful in parsing log files, web scraping or extracting structured info. from unstructured text.

iv) Search & Replace :
- Regex is used for powerful search and replace operations. It allows you to find patterns in text & replace them with new strings or modify structure of data.
- e.g. Replacing all instances of dates in a format like dd-mm-yyyy to yyyy-mm-dd.

v) Syntax Highlighting in editors :
- Many code editors & text editors use regex for syntax highlighting, identifying keywords & providing custom highlights for various prog. languages.
- e.g. Highlighting keywords in Python, such as def or class, based on patterns.

vi) Lexical Analysis in compilers :
- Regex are used in compilers & interpreters to tokenize I/p programs during lexical analysis. They help define patterns for prog. lang. syntax.
- e.g. Defining token patterns for keywords, operators & identifiers in prog. languages.

vii) Spam filtering.

viii) Natural lang. Processing (NLP).

DPU

Que.4] Determine Regular expression over alphabet $\Sigma = \{a, b\}$ for following.

i) All strings that contain an even number of b's.

⇒ Ans :

A regex for strings with an even no. of b's ensures that every occurence of b must have a matching pair. The a can appear anywhere & any no. of times.

∴ Regex = $(a^* b a^* b a^*)^*$

Here,

a^* : Allows any no. of a's (including zero) to appear before or betⁿ b's.

ba^*b : Represents a pair of b's, with any no. of a's in betⁿ.

The outermost * allows for the entire sequence to repeat, covering all possible even-numbered b strings.

ii) All strings that do not end with 'aa'.

⇒ Ans :

A regex for strings that do not end with aa ensures the either :

The string ends with b or a OR

The string ends with a sequence that is not aa.

∴ Regex = $(a+b)^* (b+ba)$

Name : Aniruddha
Roll No : TEIT-10

DPU

*** TOC : Assignment - 3 ***

Que. 1] Define the Chomsky Normal Form (CNF) Theorem.

⟹ Ans :

i) CNF stands for Chomsky Normal Form. It is a specific form of context free grammar.

ii) The CNF states that every context Free Grammar (CFG) can be converted into an equivalent grammar in CNF, except for those that generate empty language.

iii) A CFG is said to be in CNF, if all its production rules are of the following two forms :

1] $A \rightarrow BC$: where A, B, C are non-terminal symbols & B and C are not start symbols. This represents decomposition of non-terminal into exactly two non-terminals.

2] $A \rightarrow a$: where A is a non-terminal symbol and 'a' is terminal symbol. This represents the production of a terminal from non-terminal.

iii) Additionally, the start symbol S can produce the empty string ε only if grammar generates the empty language, meaning if ε is part of the language, we have a production rule like $S \rightarrow \varepsilon$ (only allowed if lang. contains ε)

iv) key-properties of CNF :

1] Binary rules :

Each production either decomposes a non-terminal into two non-terminals or produces terminal symbol.

2] No mixed productions :

No production rule can have a mixture of terminals and non-terminals on right hand side.

(e.g. rules like $A \rightarrow aB$ are not allowed).

v) CNF Theorem :

It states that,

-Any CFG can be converted to an equivalent grammar in chomsky Normal Form.

-The equivalent CFG in CNF will generate the same language (minus the empty string ε if applicable).

vi) CFG to CNF conversion :

Step-1] Eliminate ε-productions.

(Remove production rules of the form $A \to \varepsilon$)

Step-2] Eliminate unit productions.

(Remove production rules where non-terminal directly produces another non-terminal i.e. rules of form $A \to B$)

Step-3] Remove useless productions.

Step-4] Convert remaining productions.

(Ensure that all remaining production rules conform to one of the two CNF forms : either $A \to Bc$ or $A \to a$).

e.g. Consider a CFG with following production rules :

$S \to AB$

$A \to a$

$B \to Bc$

$B \to b$

$C \to b$

The conversion to CNF will give grammar like :

$S \to AB$

$A \to a$

$B \to b$

$C \to b$

DPU

Que.2] Define the two Normal Forms of CFG.

⟹ Ans :

In TOC, there are two important normal forms for context free grammars: Chomsky Normal Form (CNF) and Greibach Normal Form (GNF). Both Normal forms are used to simplify CFGs and make them easier to work with, particularly for algorithmic applications, such as parsing.

1) Chomsky Normal Form (CNF):

- A CFG is in CNF if all production rules are of following two types:
  - i) $A \rightarrow BC$ (Decomposition of non-terminals into exactly two non-terminals)
  - ii) $A \rightarrow a$ (Production of a terminal from a non-terminal)
- CNF limits each production to have either two non terminals or one terminal.
- No rules mix terminals & non-terminals on RHS.

2) Greibach Normal Form (GNF):

- A CFG is in GNF if all production rules are of the form $A \rightarrow a\alpha$: where A is non-terminal, a is terminal and α is a string of zero or more non-terminal symbols.
- In GNF, each production starts with terminal symbol, possibly followed by some non-terminal symbols.
- It is useful for top-down parsers, as it helps simplify leftmost derivations in CFG.
- e.g. For a grammar that generates the language $\{a^n b^n, n \geq 1\}$, the CFG in GNF could be:

  $S \rightarrow aA$

  $A \rightarrow bS \mid b$

Thus, both these forms helps standardize CFGs & make certain theoretical tasks easier.

DPU

Que.3] Illustrate Rightmost derivation (a+b)*c using grammar. State whether given grammar is ambiguous or not:

$E \rightarrow E+E / E*E / E / id$

⇒ Ans :

Given Grammar:-

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow id$

Thus, rightmost derivation is :-

E

$\rightarrow E*E$

$\rightarrow E*id$

$\rightarrow (E+E)*id$

$\rightarrow (E+id)*id$

$\rightarrow (id+id)*id$

$\rightarrow (a+b)*c$

- Ambiguity of Grammar :-

A grammar is ambiguous if there exists more than one distinct parse tree (or derivation) for same string.

Consider the expression : a+b*c :

∴ The expression could be parsed as :

i) (a+b)*c : Adding first & then multiplying

$E \rightarrow E*E \rightarrow (E+E)*id \rightarrow (id+id)*id \rightarrow (a+b)*c$

ii) a+(b*c) : Multiplying first & then adding.

$E \rightarrow E+E \rightarrow id+E \rightarrow id+(E*E) \rightarrow id+(id*id) \rightarrow a+(b*c)$

Since both the derivations are possible for expression a+b*c (with different parse trees representing different precedence), the grammar allows more than one parse tree for same expression, making it ambiguous.

Que.4] Show following grammar into an equivalent with no unit production & no useless symbols.

$S \rightarrow ABA$

$A \rightarrow aAA \mid aBc \mid bB$

$B \rightarrow A \mid bB \mid cB$

$C \rightarrow cC \mid cC$

⇒ Ans:

Step-1] Eliminate unit productions.

$B \rightarrow A$

Replace it by replacing A with productions of A

$\therefore B \rightarrow aAA \mid aBc \mid bB \mid bB \mid cB$

Step-2] Eliminate useless symbols

i) Eliminate non-productive symbols. (A non-terminal is productive if it can eventually derive a string of terminal symbols). — None in this case.

ii) Eliminate unreachable symbols. (A non-terminal is reachable if it can be reached from start symbol S). — None in this case.

Final Grammar:

$S \rightarrow ABA$

$A \rightarrow aAA \mid aBc \mid bB$

$B \rightarrow aAA \mid aBc \mid bB \mid cB$ ..... (Redundant production removed).

$C \rightarrow cC \mid cC$

Name : Aniruddha

Roll No: TEIT-10

DPU

✳ TOC : Assignment – 4 ✳

Que.1] Define Post Machine.

⟹ **Ans :**

i) Informally, a post machine can be viewed as finite automata with a queue. An added queue provides memory and increases the capability of machine.

ii) A Post Machine is defined as 7-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

where,

Q = Finite set of states.

Σ = Set of I/P alphabets.

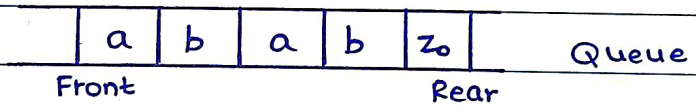
Γ = Queue symbols, $\Sigma \subseteq \Gamma$

δ = Transition function.

q_0 = Initial state.

z_0 = A special symbol, such that $z_0 \notin \Sigma$. It is used to mark rear end of I/P string initially stored in queue. An empty queue contains only symbol z_0.

F = Set of final states.



Here,

Items can be added only to rear of the queue & deleted only from the front.

A single move in Post Machine depends on :

1. The state of machine
2. Symbol at front of the queue.

iii) A Post machine is more powerful than PDA as a stack can be simulated using a queue, but a queue cannot be simulated using a stack.

DPU

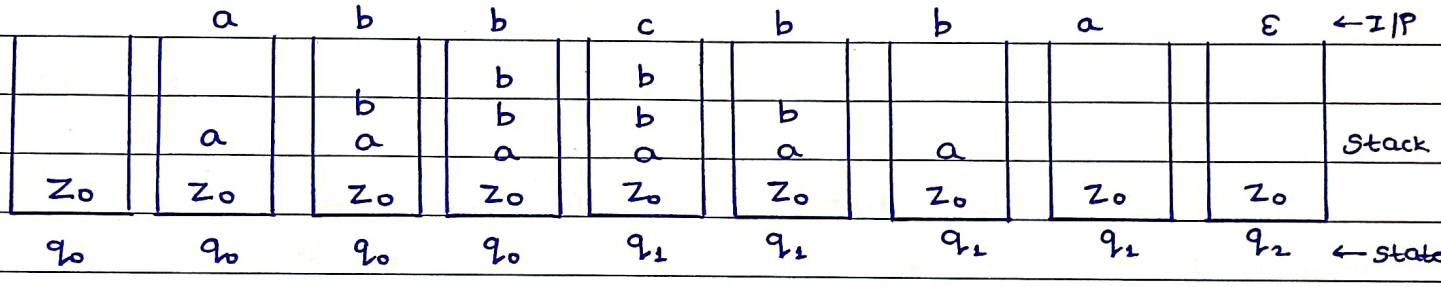
Que.2] ⇒ Design a PDA for accepting language $L=\{wcw^R \mid w \in (a,b)^*\}$

Ans :

PDA Design overview

- The PDA will push the symbols of w onto stack as it reads them.
- when it encounters middle symbol c, PDA switches to new phase.
- In second phase, for each symbol it reads from I/P (which should correspond to w^R), it pops corresponding symbol from stack.
- If all symbols are matched & stack is empty after reading entire input, PDA accepts the string.

Consider string : abbcbba.



Thus, the PDA accepting final state is given by,

$M=(\{q_0,q_1,q_2\},\{a,b,c\},\{a,b,Z_0\},\delta,q_0,Z_0,\{q_2\})$

where, transition fn (δ) is as given below :

$\delta(q_0,a,\epsilon) = (q_0,a)$ } First n symbols are pushed onto stack

$\delta(q_0,b,\epsilon) = (q_0,b)$ }

$\delta(q_0,c,\epsilon) = (q_1,\epsilon)$ — state changes on c

$\delta(q_1,a,a) = (q_1,\epsilon)$ } Last n symbols are matched with first n symbols in reverse order

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

$\delta(q_1,\epsilon,Z_0) = (q_2,Z_0)$ — Accepted the final state.

DPU

Que.3] Define Push Down Automata. Explain different types of PDA. Explain any two applications of PDA.

⟹ Ans :

i) Informally, PDA can be viewed as a Finite Automata with a stack.

A PDA uses three stack operations :

1) Push — insert symbol onto top of stack.
2) Pop — removes the top symbol from stack.
3) Nop — do nothing to stack.

ii) A PushDown Automata is defined as 7 tuple :

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

(Γ : Stack Symbols; Z_0 : Initial stack Symbol)

$\delta = Q \times (\Sigma \cup \varepsilon) \times \Gamma \rightarrow Q \times \Gamma^*$

iii) Types of PDA :-

1] Deterministic Pushdown Automata (DPDA) :

- In DPDA, for any given configuration (state, input symbol & top stack symbol), there is at most one valid transition. This means that the PDA's behaviour is fully determined at every step.
- DPDAs can only recognize a subset of CFGs.
- e.g. A DPDA can recognize languages like $L = \{a^n b^n \mid n \geq 0\}$

2] Non-deterministic Pushdown Automata (NPDA) :

- In NPDA there can be multiple possible transitions for a given configuration.
- The PDA can guess correct sequence of transitions, which makes it more powerful.
- NPDAs can recognize all CFGs.
- e.g. An NPDA can recognize languages like $L = \{a^n b^n c^n \mid n \geq 0\}$, which cannot be recognized by DPDA.

DPU

iv) Applications of PDA :-

1] Parsing in compilers :

- PDA's are used to parse the syntax of prog. languages, ensuring that the code follows correct structure.
- e.g. Parsing expressions like ((a+b)*c) in lang. like C or Java.

2] Expression Evaluation :

- PDAs are used to evaluate arithmetic expressions by recognizing context-free structures, such as operator precedence & balanced parentheses.
- e.g. Evaluating mathematical exp. like 3*(4+5).

3] XML/HTML validation :

- PDAs help validate structured documents like XML or HTML, ensuring that opening & closing tags are properly nested & matched.
- e.g. Ensuring that <html><body></body></html> is valid HTML.

ue.4] ⟹ Design a PDA for language $L = \{a^n c b^n \mid n \geq 1\}$

Ans :

$L = \{acb, aacbb, aaacbbb, \ldots\}$

For string aacbb

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

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

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

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

$\delta(q_2, b, a) = (q_2, \varepsilon)$

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

DPU

Que.5] Convert the grammar

$S \rightarrow 0S1 \mid A$

$A \rightarrow 1A0 \mid S \mid \varepsilon$

To PDA that accepts same language by empty stack.

⇒ Ans :

Step-1] For each variable $A \in V$, include a transition

$$\delta(q, \varepsilon, A) \longrightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in } G\}$$

$$\therefore \delta(q, \varepsilon, S) \longrightarrow \{(q, 0S1), (q, A)\}$$

$$\delta(q, \varepsilon, A) \longrightarrow \{(q, 1A0), (q, S), (q, \varepsilon)\}$$

Step-2] For each terminal $a \in T$, include a transition

$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$

$$\therefore \delta(q, 0, 0) \longrightarrow \{(q, \varepsilon)\}$$

$$\delta(q, 1, 1) \longrightarrow \{(q, \varepsilon)\}$$

Therefore the PDA is given by,

$M = (\{q\}, \{0,1\}, \{S, A, 0, 1\}, \delta, q, S, \phi)$

where δ is :

$\delta(q, \varepsilon, S) = \{(q, 0S1), (q, A)\}$

$\delta(q, \varepsilon, A) = \{(q, 1A0), (q, S), (q, \varepsilon)\}$

$\delta(q, 0, 0) = \{(q, \varepsilon)\}$

$\delta(q, 1, 1) = \{(q, \varepsilon)\}$

ue.6] Compare Finite Automata & Pushdown Automata.

⇒ Ans :

FA's are simpler and more limited in types of languages they can recognize, while PDA provide additional power of stack, enabling them to handle more complex structures like nested & recursive patterns found in CFG.

DPU

Finite Automata	Pushdown Automata
i) A computational model that recognizes regular languages using states & transitions.	i) A computational model that recognizes CFGs using states, transitions & stack.
ii) It has finite amount of memory. (states only)	ii) It uses a stack for memory, allowing it to store unbounded amount of info.
iii) It can recognize regular languages.	iii) It can recognize CFGs including some lang. that are not regular.
iv) Transitions depend solely on current state & I/P symbol.	iv) Transitions depend on current state, I/P symbol & top symbol of stack.
v) Accepts by reaching an accepting state.	v) Accepts by empty stack or by reaching accepting state.
vi) Simpler in terms of structure & implementation.	vi) More complex due to additional stack data structure.
vii) can be deterministic (DFA) or non-deterministic (NFA).	vii) can be deterministic (DPDA) or nondeterministic (NPDA).
viii) Less expressive; cannot handle nested structures.	viii) More expressive; can handle nested structures.
ix) e.g. $L = \{a^n, n \geq 0\}$ $L = \{0,1\}^*$	ix) e.g. $L = \{a^n b^n, n \geq 0\}$ $L = \{a^n c b^n \mid n \geq 1\}$

Name : Aniruddha

Roll No : TEIT-10

DPU

TOC - Assignment No. : 5

Que.1] Write a note on Universal Turing Machine.

⇒ Ans :

i) A Universal Turing Machine (UTM) is a crucial concept in field of TOC, representing the idea of a single machine capable of simulating any other Turing machine.

ii) A UTM is a theoretical machine that can simulate the operation of any Turing Machine on any input. It takes as input a description of another Turing Machine (along with its input) & behaves as if it were that machine.

iii) Components of UTM :

1] Tape : An infinite tape divided into cells that can hold symbols from a finite alphabet. The tape serves as both input & workspace for computation.

2] Head : A read/write head that can move left or right along tape, reading symbols & writing new ones.

3] State Register : keeps track of the current state of machine, which influences its behaviour.

4] Transition Fn : A set of rules that dictate how the machine transitions between states based on current state & symbol being read.

iv) Operation :

1) Input : The UTM takes two inputs :

- An encoded description of Turing machine M (including its states, alphabet & transition rules).
- An I/P string w on which M is to operate.

2) Simulation : The UTM decodes the description of M and simulates its transitions step by step, processing I/P string w as M would.

v) Importance :

1) computability.

2) complexity Theory.

3) Language Recognition.

DPU

Que.2] Construct a Turing Machine to find 2's complement of a binary number.

⟹ Ans :

i) 2's complement of a binary number can be found by not changing bits from right end till the first '1' and then complementing remaining bits.

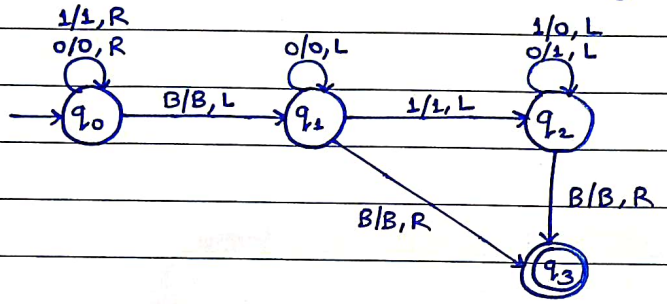
e.g. 2's complement of 0101101000 = 1010011000

ii) Algorithm:

1) Locate the last bit (rightmost)

2) Move towards left till the first 1

3) complement remaining bits, while moving towards left.



	0	1	B	
→ q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, B, L)	
q_1	$(q_1, 0, L)$	$(q_2, 1, L)$	(q_3, B, R)	
q_2	$(q_2, 1, L)$	$(q_2, 0, L)$	(q_3, B, R)	
q_3^*	q_3	q_3	q_3	← Halting state

The Turing machine M is given by:

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

where,

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

B = Blank Symbol

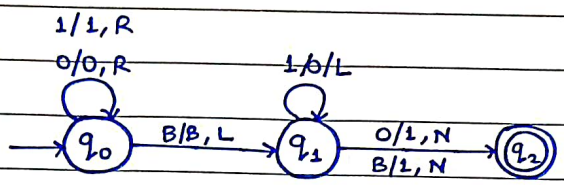
DPU

Que.3] Design a Turing machine to increment a value of binary no. by 1.

⇒ Ans :

To increment a binary number by 1, following steps are required:

1) Trailing 1's should become 0.
2) First '0' from right end should become 1.
3) Remaining bits will remain as it is.



	0	1	B	
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, B, L)	
q_2	$(q_2, 1, N)$	$(q_1, 0, L)$	$(q_2, 1, N)$	
q_2*	q_2	q_2	q_2	← Halting state

- State q_0 moves the head to first blank on right side.
- A transition from q_0 to q_1, positions the head on rightmost 0 or 1.
- Trailing 1's are written as 0's in state q_1.
- A Transition from q_1 to q_2 changes the first '0' or a blank from the right end towards left to 1.
- q_2 is halting state.

Que.4] Write short notes on :

1) Unsolvable Problems:

⇒ Ans :

i) They are decision problems for which no algorithm can provide a soln for all possible I/Ps.

ii) In other words, there is no Turing Machine that can always halt and give the correct answer (yes/no) for every I/P.

iii) Examples:

a] Halting Problem: Given a description of a Turing machine & an input, determine whether the machine will eventually halt or run forever.

b] Decision Problems in Logic: Problems like Truth of arbitrary logical statements can be undecidable in certain logical systems.

iv) The study of unsolvable problems is fundamental in computability theory as it defines the limits of what can be computed or solved algorithmically.

2) Applications of Turing Machine:

⇒ Ans:

i) Theory of computation: Turing Machines serve as a foundational model for studying computability & complexity. They help formalize the concept of algo. & what it means for a problem to be computable.

ii) Language Recognition: Turing machines can be used to recognize & decide various classes of formal languages, including context free & recursively enumerable languages.

iii) Software verification: TM's are used in formal verification processes to prove the correctness of s/w.

iv) Artificial Intelligence: Turing Machines provide a theoretical basis for understanding ML & AI.

v) Complexity classes: Turing machines helps define various complexity classes (P, NP, NP-complete), facilitating the classification of the computational problems based on their resource requirement & solvability.

DPU

Que.5] What are recursive and recursively enumerable languages?

⇒ Ans :

i) Recursive Languages:

- A language L is called recursive (or decidable) if there exists a Turing Machine M that decides L. This means that M will accept any string in the language (halt and return 'yes') & will reject any string not in the language (halt and return 'no').
- It is guaranteed to have a TM that always halts on all I/Ps.
- e.g.

The set of all strings that are well formed arithmetic expressions.

ii) Recursively Enumerable Languages:

- A language L is called recursively enumerable (or semi-decidable / acceptable) if there exists a Turing Machine M that can recognize L. This means that M will accept any string in the language but may either reject or run forever (not halt) for strings not in language.
- It may have TM that halts only when I/P is in the language.
- e.g.

The set of all valid proofs in formal system.

Name: Aniruddha
Roll No.: TE17-10

DPU

*** TOC : Assignment – 6 ***

Que.1] Write a note on computational complexity.

⇒ Ans :

i) Computational complexity is a branch of theoretical comp. science that studies the resources required to solve computational problems, particularly time & space.

ii) It focuses on classifying the problems based on their inherent difficulty & efficiency of algorithms that solve them.

iii) Complexity Measures :

1] Time complexity : The amount of time (usually measured as no. of steps) an algo. takes to solve a problem as a fn of I/P size.

2] Space complexity : The amount of memory an algorithm uses during its execution.

3] Input size : The size of I/P affects how computational resource grow. Larger inputs typically require more time & space to process.

iv) Complexity classes :

It groups problems based on resources required to solve them :

1] P (Polynomial Time) : The class of problems that can be solved in polynomial time by deterministic TM. These are considered tractable or efficiently solvable.

e.g. sorting an array using Merge sort ($O(n \log n)$).

2] NP (Nondeterministic Polynomial Time) : The class of problems for which a proposed soln can be verified in polynomial time by a deterministic TM.

e.g. The Travelling Salesman Problem.

3] P v/s NP Problem : A major open que. in CS asks whether P=NP i.e. whether every problem whose soln can be verified in polynomial time can also be solved in polynomial time.

DPU

4] NP-complete : A subset of NP problems that are most difficult ; if any NP-complete problem can be solved in polynomial time, all NP problems can be.

5] NP-Hard : Problems that are at least as hard as NP-complete problems but may not belong to NP.

Que.2] Explain SAT Problem in brief.

⟹ Ans :

i) In TOC, The SAT (Boolean Satisfiability Problem) plays a crucial role in understanding the limits of what can be computed efficiently.

ii) The SAT problem asks whether, for a given Boolean formula composed of variables & logical operations (AND, OR, NOT), there exists an assignment of Truth values (True/False) to the variables that makes the entire formula evaluate to True. If such an assignment exists, the formula is said to be satisfiable ; otherwise, it is unsatisfiable.

iii) e.g.

Consider the Boolean Formula :

$F = (x_1 \vee \sim x_2) \wedge (x_2 \vee x_3) \wedge (\sim x_1 \vee \sim x_3)$

We need to find a truth assignment for variables x_1, x_2 & x_3 that makes the formula true.

One possible satisfying assignment is $x_1 = T$, $x_2 = T$ & $x_3 = F$.

iv) Importance in TOC :

1] NP-completeness :-

- SAT was first problem proven to be NP-complete
- This means SAT is as hard as any problem in NP.

2] Reductions :-

- Many other computational problems (graph coloring, scheduling, ...) can be transformed into instances of SAT problem.
- This makes SAT a central problem in complexity theory & algo. design.

DPU

Que.3] Explain Node cover problem with example.

⇒ Ans :

i) The Node cover problem (also known as vertex cover Problem) is a classic problem in graph theory & computer science.

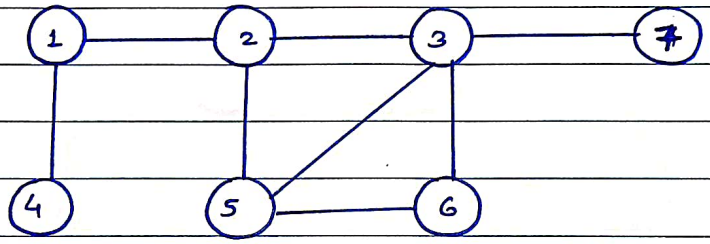
ii) It involves finding a subset of vertices (or nodes) in a graph such that every edge in graph is connected to at least one of the selected vertices.

iii) Defination :

Given an undirected graph $G=(V,E)$, where V is the set of vertices & E is set of edges, a vertex cover is a subset $V' \subseteq V$ such that for every edge $(u,v) \in E$, at least one of u or v belongs to V'.

iv) Example

Consider given graph.



Initially,

Find degree of all nodes & delete edges associated with the nodes having maximum degree.

1 (2,4) → 2

2 (1,5,3) → 3

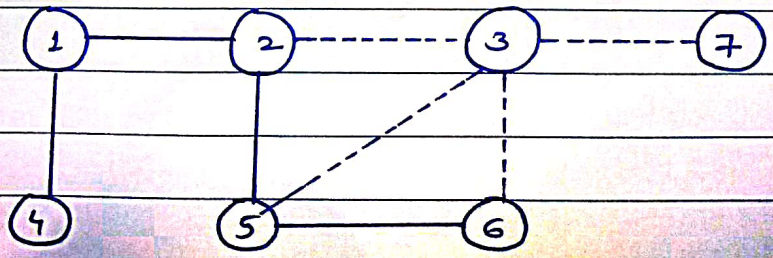
3 (2,5,6,7) → 4 ⇐ Delete edges associated with node 3

4 (1) → 1

5 (2,3,6) → 3

6 (3,5) → 2

7 (3) → 1



Now,

1(2,4) → 2

2(1,5) → 2

3(0) → 0

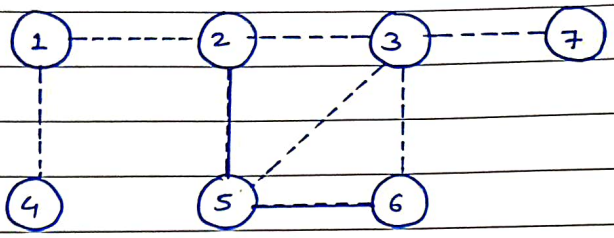
4(1) → 1

5(2,6) → 2

6(5) → 1

7(0) → 0

Now, delete edges associated with nodes 1, 2 & 5.



∴ Finally we get degree of all nodes as zero.

Thus, {3, 1, 5} is the valid vertex cover with size 3.

problem : A major open que. in cs asks