TCS Theory Answer bank

1.Basic concepts and Finite Automata

1. Explain applications of Finite Automata(FA).

Finite automata are used for solving several common types of computer algorithms. Some of them are :

(i) Design of digital circuit

(ii) String matching

(iii) Communication protocols for information exchange.

(iv) Lexical analysis phase of a compiler.

Finite automata can work as an algorithm for regular language. It can be used for checking whether a string w∈L, where L is a regular language.

v. Text Search and Pattern Matching
vi. Spell Checkers
vii. Regular Expression Matching
viii. Natural Language Processing (NLP)

2. Discuss difference in transition function of FA, PDA and TM.

Differences in Transition Functions

Aspect	Finite Automaton (FA)	Pushdown Automaton (PDA)	Turing Machine (TM)
Form of Transition Function	$\delta : Q \times \Sigma \to Q$	$\delta : Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$	$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$
Inputs	Current state and input symbol	Current state, input symbol, and top of stack symbol	Current state and tape symbol
Outputs	Next state	Next state and stack operation (push, pop, or no change)	Next state, symbol to write on tape, and head movement (left, right, or stay)
Memory Used	No memory (only states)	Stack memory (last-in-first-out, or LIFO)	Infinite tape (acts as read/write memory)
Operations	State transition based on input symbol	State transition, stack operations based on input symbol and stack top	State transition, tape read/write, and head movement
Example Transition	$\delta(q_0, a) = q_1$	$\delta(q_0, a, X) = (q_1, \gamma)$, where X is popped and γ is pushed	$\delta(q_0, a) = (q_1, b, R)$, where a is overwritten by b and head moves right

3. Explain applications of Regular Expressions.

1. Lexical Analysis in Compiler Design

- Lexical analysis is the first phase of a compiler, where source code is broken down into tokens, Regular expressions play a critical role in defining the patterns for tokens.

2. grep in UNIX

- The grep command in UNIX is a powerful tool that uses regular expressions to search text in files or streams. Here are some common applications of grep:

3. Input Validation:

- Regular expressions are commonly used to validate user input in forms and applications to ensure it meets certain criteria.

4. Data Extraction:

- Regex can extract specific information from text by identifying patterns.

5. Search and Replace Operations:

- Regular expressions make search-and-replace operations more flexible, allowing bulk modifications based on patterns rather than exact matches.

4. Define Regular language.

A **Regular Language** is a type of formal language that can be recognized by a **finite automaton** and can be defined using **regular expressions**. In simpler terms, it is a language whose strings can be generated by applying a specific set of rules, such as concatenation, union, and repetition (also known as the Kleene star).

Examples of Regular Languages

- The set of all strings over the alphabet {0,1} that contain an even number of 0s.
- The set of all strings over {a,b} that start with a and end with b.

5. Short note on Arden's theorem.

Arden's Theorem is a fundamental theorem in the theory of regular languages and automata, used to solve regular expressions for certain types of equations. It provides a method to express the language accepted by a **finite automaton** in terms of regular expressions, which is useful in designing and analyzing automata.

Statement of Arden's Theorem

For any two regular expressions P and Q over an alphabet Σ if R is a solution to the equation:

$R=Q+RP$

then the solution can be expressed as:

$R=QP*$

where:

- + represents the union of languages,
- · represents concatenation,
- $P*$ is the Kleene star of P, meaning zero or more occurrences of P.

Q.2. Write note on Chomsky Hierarchy.

(MU Dec. 2009, Dec. 2012, May 2013, May 2014, Dec. 2014, May 2015, Dec. 2016, May 2017, Dec. 2017)

Ans: Chomsky Hierarchy

A grammar can be classified on the basis of production rules. Chomsky classified grammars into the following types:

1. Type 3: Regular grammar

2. Type 2: Context free grammar

3. Type 1: Context sensitive grammar

4. Type 0: Unrestricted grammar.

No	Grammar Type	Grammar Accepted	Language Accepted	Automation
1	Type 3	Regular grammar	Regular Language	Finite Automata (FA)
2	Type 2	Context free grammar (CFG)	Context free Language (CFL)	Pushdown Automata (PDA)
3	Type 1	Context sensitive grammar (CSG)	Context sensitive Language (CSL)	Linear Bounded Automation (LBA)
4	Type 0	Unrestricted Grammar	Recursively enumerable language (REL)	Turing Machine (TM)

1. Type 3 or Regular Grammar

A grammar is called Type 3 or regular grammar if all its productions are of the following forms:

$A \rightarrow \varepsilon$

$A \rightarrow a$

$A \rightarrow aB$

$A \rightarrow Ba$

Where, $a \in \Sigma$ and $A, B \in V$.

A language generated by Type 3 grammar is known as regular language.

2. Type 2 or Context Free Grammar

A grammar is called Type 2 or context free grammar if all its productions are of the following form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (VUT)^*$.

V is a set of variables and T is a set of terminals.

The language generated by a Type 2 grammar is called a context free language, a regular language but not the reverse.

3. Type 1 or Context Sensitive Grammar

A grammar is called a Type 1 or context sensitive grammar if all its productions are of the following form.

$\alpha \rightarrow ß$

Where, ß is atleast as long as α.

4. Type 0 or Unrestricted Grammar

Productions can be written without any restriction in a unrestricted grammar. If there is production of the $\alpha \rightarrow ß$, then length of α could be more than length of ß.

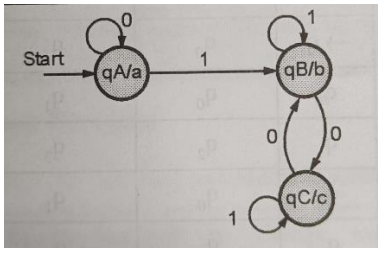
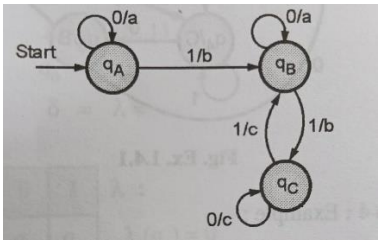
Every grammar also is a Type 0 grammar.

A Type 2 grammar is also a Type 1 grammar

A Type 3 grammar is also a Type 2 grammar.

Q.3. Difference between moore machine and mealy machine

Ans:

No	Moore Machine	Mealy Machine
1	In moore machine, output symbol is associated with each state	In mealy machine, output symbol is associated with each transition
2	Output is dependent on state	Output is dependent on state and input
3	Output Mapping: $\lambda : Q \rightarrow \Delta$	Output Mapping: $\lambda : Q \times \sum \rightarrow \Delta$
4	In this, if length of input sequence is n, then the length of output sequence is n+1	In this, if length of input sequence is n, then the length of output sequence is also n
5	Here we can get output on €	Here we cannot get output on €
6	Example:  	Example: 

Q.4. Difference between NFA and DFA:

	DFA	NFA
1	DFA stands for Deterministic Finite Automata.	NFA stands for Nondeterministic Finite Automata.
2	DFA cannot use Empty String transition.	NFA can use Empty String transition.
3	DFA is more difficult to construct.	NFA is easier to construct.
4	Time needed for executing an input string is less.	Time needed for executing an input string is more.
5	All DFA are NFA.	Not all NFA are DFA.
6	DFA requires more space.	NFA requires less space then DFA.
7	Dead state may be required.	Dead state is not required.
8	Backtracking is allowed in DFA.	Backtracking is not always possible in NFA.
9	Conversion of Regular expression to DFA is difficult.	Conversion of Regular expression to NFA is simpler compared to DFA.

Q.5. Write Short Note On: Recursive and Recursively Enumerable Languages

(May 14,Dec 16) **OR**

Difference between Recursive and Recursively Enumerable Languages:

Comparison	Recursive Language	Recursively enumerable language
Also Known as	Turing decidable languages	Turing recognizable languages
Definition	In Recursive Languages, the Turing machine accepts all valid strings that are part of the language and rejects all the strings that are not part of a given language **and halt.**	In Recursively enumerable languages, the Turing machine accepts all valid strings that are part of the language and rejects all the strings that are not part of the given language **but do not halt and starts an infinite loop.**
States	(1) Halt and accept (2) Halt and Reject	(3) Halt and accept (4) Halt and Reject (5) Never Halt (Infinite loop)
Loop	Finite Loop	Infinite loops of machine are possible
Halting	Halting Turing Machine	Non Halting Turing Machine
Accept/ Reject	Accept (Turing machine) = L Reject (Turing machine) = L' Loop (Turing machine) = φφ φ = null φ = null	Accept (Turing machine) = LReject (Turing machine) + Loop (Turing machine) = L'
Closed under	Closed under all except homomorphism, substitution, GSM mapping, and rational transduction	Closed under all except set difference, and complementation.
	context-sensitive language	RE languages or type-0 language

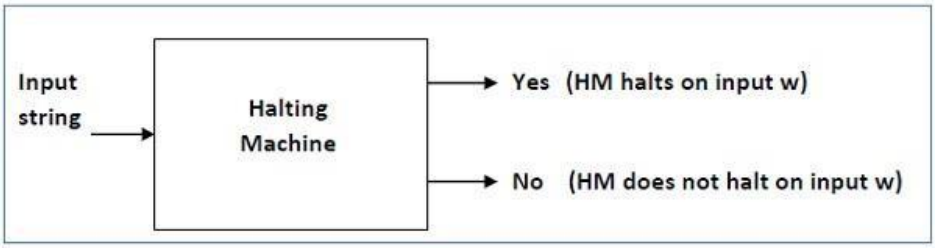
Q.6. State and prove Halting Problem or Short note on Halting Problem (Dec15, Dec16)

Ans:

Input – A Turing machine and an input string w.

Problem – Does the Turing machine finish computing of the string w in a finite number of steps? The answer must be either yes or no.

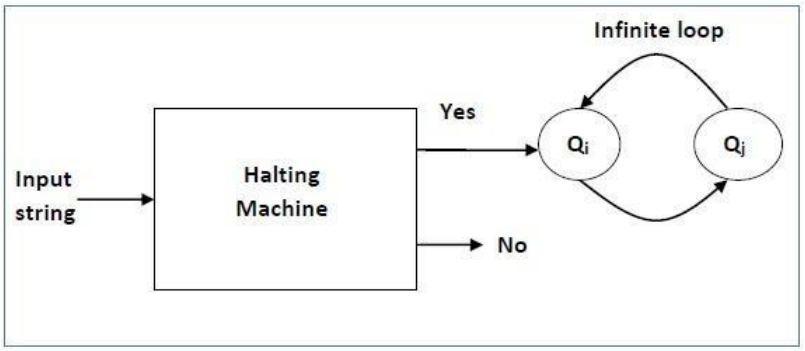
Proof – At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a Halting machine that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine:



Now we will design an inverted halting machine (HM)' as –

- If H returns YES, then loop forever.
- If H returns NO, then halt.

The following is the block diagram of an 'Inverted halting machine':



Further, a machine (HM)2 which input itself is constructed as follows –

- If (HM)2 halts on input, loop forever.
- Else, halt.

Here, we have got a contradiction. Hence, the halting problem is undecidable.

Q.7. Give formal definition of PDA / Mathematical Model of PDA (Dec16)

Ans:

i) PDA is used for recognizing context free language which is generated by context free grammar.

ii) PDA is more powerful than finite automata because it has a stack which can be used for remembering some information.

iii) Model of PDA is as follows:

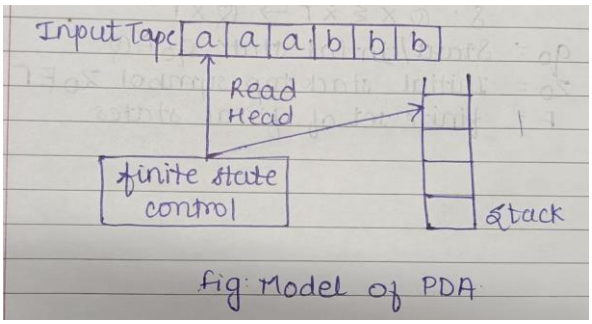


fig: Model of PDA

iv) PDA consists of finite set of states, i/p tape, read head and stack.

v) Depending on the stack, i/p symbol and stack top symbol:
- PDA can change the stack/remain in the same stack.
- PDA moves the head to right of current cell.
- PDA can perform some stack operations.

vi) PDA can be represented mathematically as follows: (7 tuples)

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

Q: the finite set of states

$\sum$: input alphabet

Γ: stack alphabet

q_0 : initial state

Z_0: initial stack top symbol

F: finite set of final states

δ: transition function

$\delta: Q \times \sum \times \Gamma \rightarrow Q \times \Gamma^*$

Q.8. State and explain power and limitation of turing machine. (Dec16)

Ans:

Power of Turing Machine:

- The **turing machine has a great computational capabilities**. So it can be used as a general mathematical model for modern computers.
- **Turing machine can model even recursively enumerable languages**. Thus the **advantage of turing machine is that it can model all the computable functions as well as the languages for which the algorithm is possible.**

Limitations of turing machine:

- Computer scientists believe that a Turing machine encapsulates the idea of computability. That is, if a function can be computed by any physical process, it can be computed by a Turing machine. (This is known as the Church-Turing thesis.)
- But we do know that **many important functions are uncomputable. For example:**
  - **Determining if a program will ever halt on a given input**
  - **Determining if two programs compute the same output**
  - **Determining the size of the smallest program that computes a given** output (formally, this is known as Kolmogorov complexity)

Since none of these can be computed by a Turing machine, we believe that they are uncomputable under any realizable model of computation.

Q.9. Write Short Note on Universal Turing Machine (Dec15)

Ans:

- Universal Turning Machine stimulates a Turning Machine.
- Universal Turing Machine can be considered as a subset of all the Turing machines, it can match or surpass other Turing machines including itself.
- Programmable Turing Machine is called Universal Turing Machine
- Universal Turing Machine is like a single Turing Machine that has a solution to all problems that is computable.
- It minimizes space complexity
- It contains a Turning Machine description as input along with an input string, runs the Turning Machine on the input and returns a result.
- The transition function is $Q \times T \rightarrow Q \times T \times \{L, R\}$, where Q is a finite set of states, T is the tape of the alphabet

Q.10. Difference between pushdown automata and finite automata

Ans:

S.NO	Pushdown automata	finite automata
1.	For Type-2 grammar we can design pushdown automata.	For Type-3 grammar we can design finite automata.
2.	Non – Deterministic pushdown automata has more powerful than Deterministic pushdown automata.	Non-Deterministic Finite Automata has same powers as in Deterministic Finite Automata.
3.	Not every Non-Deterministic pushdown automata is transformed into its equivalent Deterministic pushdown Automata.	Every Non-Deterministic Finite Automata is transformed into an equivalent Deterministic Finite Automata
4.	Context free languages can be recognized by pushdown automata.	Regular languages can be recognized by finite automata.
5.	Pushdown automata has the additional stack for storing long sequence of alphabets.	Finite Automata doesn't has any space to store input alphabets.
6.	It gives acceptance of input alphabhets by going up to empty stack and final states.	It accepts the input alphabets by going up to final states.

OR

S.NO	Pushdown automata	finite automata
1.	PDA is more powerful than FA	FA is less powerful as compare to PDA
2.	It has additional memory in form of stack	It has no memory
3.	PDA accepts CFG(Context Free Grammar)	FA accepts RL(Regular Language)
4.	PDA is useful for parsing phase of compiler	FA is useful for lexical analysis phase of compiler
5.	PDA transition based on current input or current stack on the top of stack symbol	FA transition based on current input and current state.

Q.11. Describe Finite Automata (dec18)

Ans:

→Finite Automata (FA) is the simplest machine to recognize patterns.

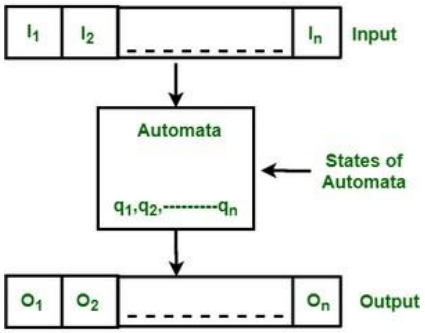
→The finite automata or finite state machine is an abstract machine that has five elements or tuples.

→ A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

1. Q: finite set of states
2. ∑: finite set of the input symbol
3. q0: initial state
4. F: **final** state
5. δ: Transition function

→It has a set of states and rules for moving from one state to another but it depends upon the applied input symbol.

→ The following figure shows some essential features of general automation.



→The above figure shows the following features of automata:

- Input
- Output
- States of automata
- State relation
- Output relation

→FA is characterized into two types:

1) Deterministic Finite Automata (DFA)
2) Nondeterministic Finite Automata(NFA)

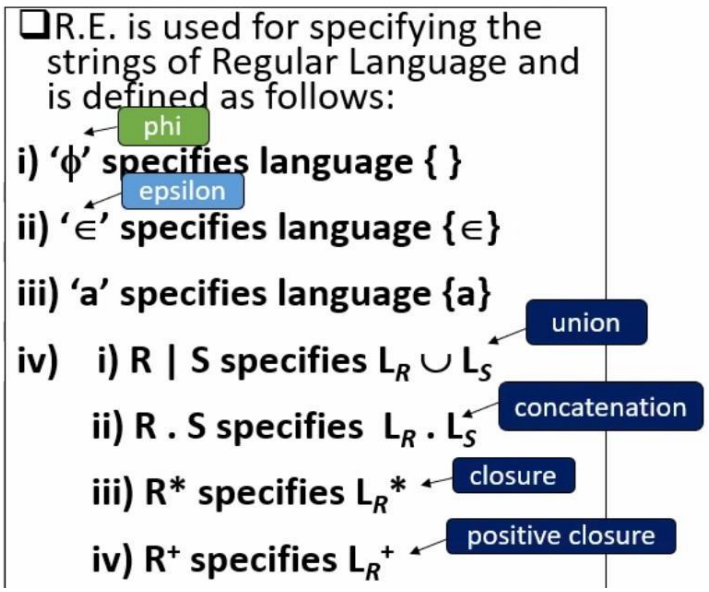
Q.12. Difference between DPDA and NPDA (explain npda – dec 18)

Ans:

S. No	DPDA (Deterministic Pushdown Automata)	NDPA (Non-deterministic Pushdown Automata)
1.	It is less powerful than NPDA.	It is more powerful than DPDA.
2.	It is possible to convert every DPDA to a corresponding NPDA.	It is not possible to convert every NPDA to a corresponding DPDA.
3.	The language accepted by DPDA is a subset of the language accepted by NDPA.	The language accepted by NPDA is not a subset of the language accepted by DPDA.
4.	The language accepted by DPDA is called DCFL (Deterministic Context-free Language) which is a subset of NCFL (Non-deterministic Context-free Language) accepted by NPDA.	The language accepted by NPDA is called NCFL (Non-deterministic Context-free Language).

Q. 16. Give formal definition of regular language. (Every year they ask this question)

Ans:



Q.17. Variants or Variations or Types of Turing Machine (Very IMP)

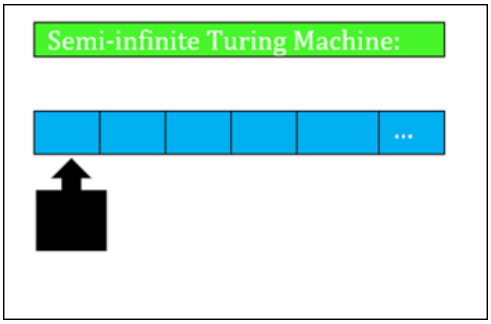
Ans:

Variants of Turing Machine are as follows:

1) **Semi-infinite turing machine:**

→It has no cells on the left-hand side of the initial position and infinite cells on the right-hand side of the initial position.

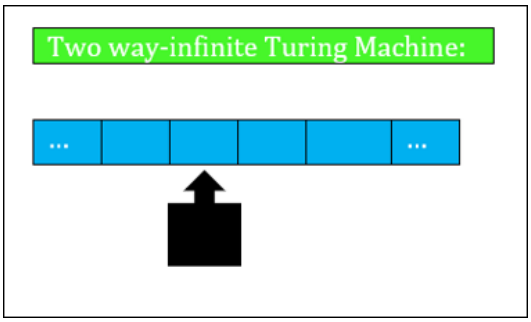
→Here, head can only be move or is allowed to move in right hand side direction of the initial position of the input on the tape.



2) **Two way-infinite turing machine:**

→The input and output tape is a two way indefinite tape i.e there are unlimited blank cells on the left as well as on the right of the current non block portion on the tape

→Two way-infinite tape turing machine can be stimulated by Standard turing machine(one way-infinite turing machine)

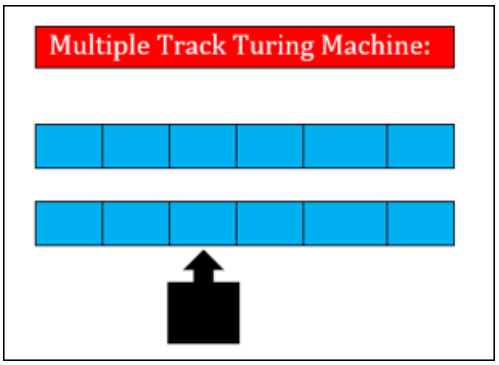


3) **Multiple track turing machine:**

→ This turing machine has multiple tracks but a single tape head that reads and writes on all tracks.

→ Also, for every single-track turing machine, there exists an equivalent multi-track turing machine.

→ Multiple track turing machine can be stimulated by Standard turing machine.

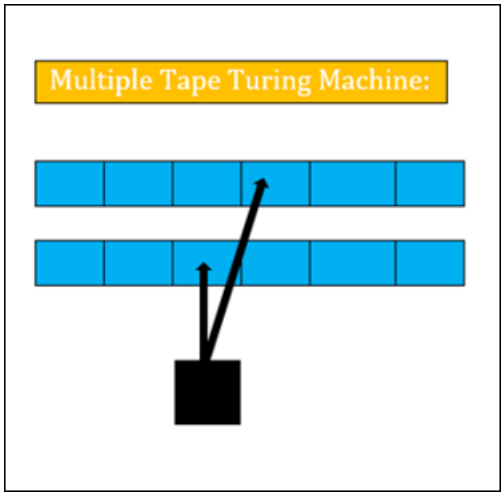


4) Multiple tape turing machine:

→It has multiple tapes and is controlled by a single head.

→Multiple-tape turing machine is different from multi track turing machine but expressive power is same.

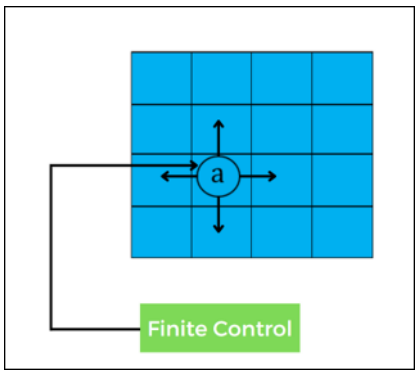
→ Multiple-tape turing machine is simulated by single-tape turing machine.



5) Multi Dimensional Turing Machine:

→It has multi dimensional tape where head can move in any direction that is left, right, up and down.

→ Multi dimensional turing machine can be stimulated by Standard turing machine.



6) Non deterministic turing machine:

→A non-deterministic Turing machine has a single, one-way infinite tape.

→For a given state and input symbol has at least one choice to move (finite number of choices for the next move), each choice has several choices of the path that it might follow for a given input string.

→A non-deterministic Turing machine is equivalent to the deterministic Turing machine.

Q.18. Closure properties of regular language (State and explain any 5 --- 5 marks)

Ans:

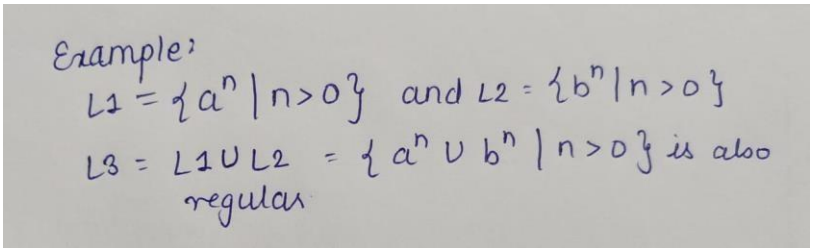
Closure properties on regular languages are defined as certain operations on regular language which are guaranteed to produce regular language.

Closure properties of regular languages are as follows:

1. Union
2. Intersection
3. Concatenation
4. Kleen closure
5. Complement
6. Reversal
7. Let difference operator
8. Homomorphism
9. Inverse homomorphism

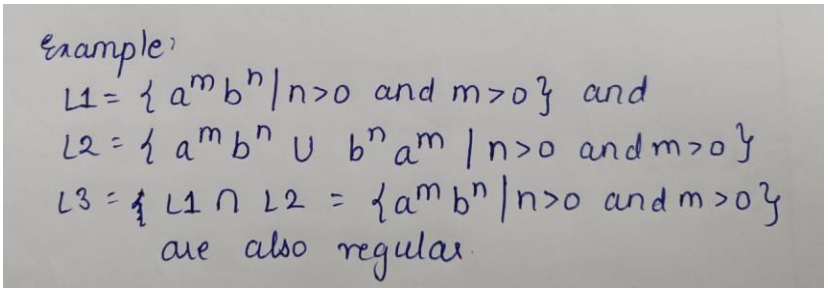
1. Union:

If L1 and If L2 are two regular languages, their union L1 U L2 will also be regular.



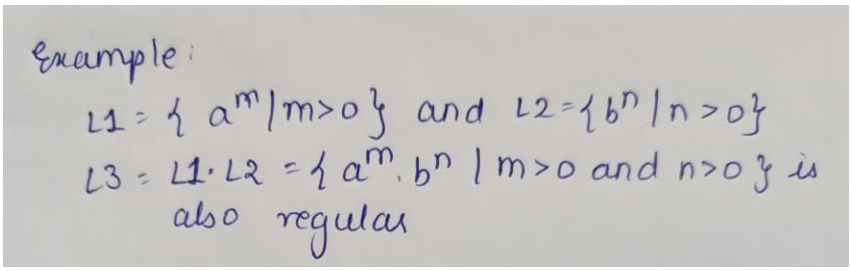
2. Intersection:

If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular.



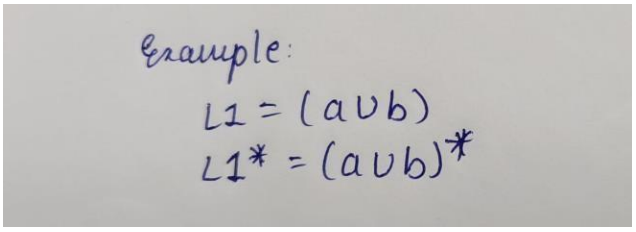
3. Concatenation:

If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular.



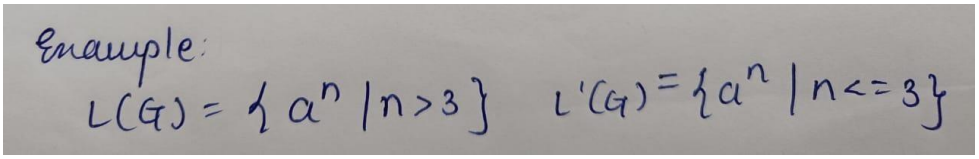
4. Kleen closure:

If L1 is a regular language, its Kleene closure L1* will also be regular.



5. Complement:

If L(G) is a regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings.



Q.19. Decision properties of regular language

Ans:

Testing Emptiness of Regular Languages:

- The emptiness question is whether there is any path from the start state to some accepting state. If so, the language is non empty, while if accepting states are all separated from the start state, then the language is empty.
- Deciding whether we can reach an accepting state from the start state is a simple instance of graph reachability.

Testing Membership in a Regular Language:

- Here the question is given a string w and a regular language L, is w in L.
- If L is represented by a DFA, simulate the DFA processing the string input symbols w, beginning in the start state. If the DFA ends in an accepting state, the answer is yes otherwise the answer is no.
- If L has any other representation besides a DFA we could convert to a DFA and then run the test above.

Testing Equivalence of States:

- To find states that are equivalent, we try to find pairs of states that are distinguishable.Any pair of states that we do not find distinguishable are equivalent.
- The table filling algorithm is a recursive discovery of distinguishable pairs in a DFA.

Testing Equivalence of Regular Languages:

- The table filling algorithm gives us an easy way to test if two regular languages are thesame. Suppose L and M are two languages represented by their respective DFA's.
- Now test if the start states of the two original DFA's are equivalent, using the tablefilling algorithm. If they are equivalent, then L = M and if not then L ≠ M.

Q.20. Post Correspondence Problem

Ans:

Post's Correspondence Problem (PCP) was first introduced by Emil Post in 1946. Later, the problem was found to have many applications in the theory of formal languages.

The problem over an alphabet Σ belongs to a class of yes / no problems and is stated as follows :

> Consider the two lists $x = (x_1 \ldots . x_n)$, $y = (y_1 \ldots\ldots y_n)$ of nonempty strings over an alphabet $\Sigma = (0, 1)$. The PCP is to determine whether or not there exist $i_1 . \ldots ., i_m$, where $o \leq i_j \leq n$, such that $x_i 1, \ldots . xi_m = yi_1, \ldots\ldots yi_m$

Note :

1) The indices i_j's need not be distinct and m may be greater than n.
2) if there exists a solution to PCP, there exists infinitely many solutions
3) PCP is undecidable.

4. **Write a short note on: Recursive and Recursively enumerable languages.**

Recursive Languages

- **Definition**: A language is **recursive** (or **decidable**) if there exists a Turing Machine that can always determine, in a finite amount of time, whether any given string belongs to the language (accepts) or does not belong (rejects).
- **Properties**:
  - For a recursive language, the Turing Machine will always halt with a definitive answer (accept or reject) for any input.
  - Recursive languages are the class of languages that can be fully "decided" by a Turing Machine.
- **Example**: The set of all even numbers in binary form is recursive since there is a clear algorithm (ending in zero) to determine if a binary number is even.

Recursively Enumerable (RE) Languages

- **Definition**: A language is **recursively enumerable (RE)** if there exists a Turing Machine that will accept any string in the language but may either reject or run indefinitely for strings not in the language.
- **Properties**:
  - For an RE language, the Turing Machine is guaranteed to halt if the string is in the language (accept), but it may never halt if the string is not in the language.
  - RE languages are also known as **semi-decidable** because the Turing Machine may not provide a definitive answer (halt) for strings not in the language.
- **Example**: The **halting problem** (determining if a given Turing Machine halts on a given input) is RE because, if the machine halts, there's a way to verify it, but if it doesn't halt, there's no guaranteed way to detect non-halting.

3. Write a short note on: Rice's theorem.

Rice's Theorem is an important result in the theory of computation, which states that **any non-trivial property of the language recognized by a Turing Machine is undecidable**.

Key Points of Rice's Theorem

1. **Non-trivial Property**: A property is considered non-trivial if it is true for some Turing Machines and false for others. In other words, the property does not apply universally to all TMs or none at all.
2. **Undecidability**: Rice's Theorem shows that it is impossible to design an algorithm (or Turing Machine) that can determine any non-trivial property of the language a Turing Machine recognizes. This means that for properties like "the language recognized by a TM is empty," "the language is finite," or "the language includes a specific string," no algorithm can decide them for all possible Turing Machines.
3. **Scope of the Theorem**: Rice's Theorem applies to properties of the **language recognized** by a Turing Machine, not properties of the machine itself (like the number of states or transitions).

Examples of Properties Covered by Rice's Theorem

- Whether a Turing Machine accepts all strings.
- Whether a Turing Machine's language is regular or context-free.

3. **Explain applications for PDA.**

1) **Syntax Parsing in Compilers:** PDAs are used to parse context-free grammars, which helps in analyzing the syntax of programming languages to check if code is written correctly.
2) **Language Recognition:** PDAs can recognize context-free languages, like checking if parentheses are balanced in an expression (e.g., in mathematical equations or code blocks).
3) **Natural Language Processing (NLP):** PDAs help model simple structures in human languages, such as basic sentence structures and nested phrases, which aids in understanding and processing human languages.
4) **XML Parsing**: XML documents have a nested structure that can be validated using PDAs, ensuring the correct opening and closing of tags.
5) **Arithmetic Expression Evaluation:** PDAs can be used to evaluate arithmetic expressions by parsing and evaluating expressions with nested structures, such as ((2+3)*4).
6) **Design of Interactive Systems:** PDAs model systems with nested or recursive states, like navigation systems with menus and submenus.