AT Theory Questions

1. What is Moore machine & Mealy machine? Describe with all tuples in detail. Differentiate between Moore machine & Mealy machine.

- Moore machine:
  - Moore machines are finite state machines with output value & its output depends only on present input.
  - Defined as: $(Q, q_0, \Sigma, O, \delta, \lambda)$
    where,
    Q – finite set of states
    q_0 – initial state
    Σ – input alphabet
    O – output alphabet
    δ – transition function which maps $Q \times \Sigma \rightarrow Q$
    λ – output function which maps $Q \rightarrow O$
- Mealy machine:
  - Mealy machines are finite state machines with output value associated with transition & its output depends only on present input.
  - Defined as: $(Q, q_0, \Sigma, O, \delta, \lambda')$
    where,
    Q – finite set of states
    q_0 – initial state
    Σ – input alphabet
    O – output alphabet
    δ – transition function which maps $Q \times \Sigma \rightarrow Q$
    λ' – output function which maps $Q \times \Sigma \rightarrow O$

Moore machine	Mealy machine
Definition - $(Q, q_0, \Sigma, O, \delta, \lambda)$	Definition - $(Q, q_0, \Sigma, O, \delta, \lambda')$
Output is associated with state.	Output is associated with transition
As each state contains an output associated to it, the number of states in machine is more compared to Mealy	Requires less states as output is associated with transition
When machine is turned on, output associated with initial state is obtained without giving an input	Output is obtained only when an input causes the transition
The first character/prefix of the output (upon initialization) must be ignored	No extra character in the output
State changes are more frequent	State changes are less frequent
Ex: 1's complement (Moore machine)	Ex: 1's complement (Mealy machine)

2. What is meant by ambiguous grammar?

- A Context-Free Grammar (CFG) is called ambiguous if there is a string that can have more than one valid derivation & valid parse tree.
- The string can be generated in different ways, either through different Left Most Derivations (LMDT) or Right Most Derivations (RMDT)
- CFG is defined by $G = (V, T, P, S)$, it is ambiguous if there is a string in terminal set T that can be produced in more than one way, creating multiple parse trees.
  V – set of variables (non-terminal symbols)
  T – set of terminal symbols
  P – set of production rules
  S – start symbol (special variable from where the derivation begins)
- Ambiguity also happens when a string has multiple possible ways to be derived using same grammar rules, leading to more than one valid tree structure. This means even if a grammar does not have both left & right derivation/recursion, it can still be ambiguous. The absence of recursion does not guarantee that the grammar is unambiguous.

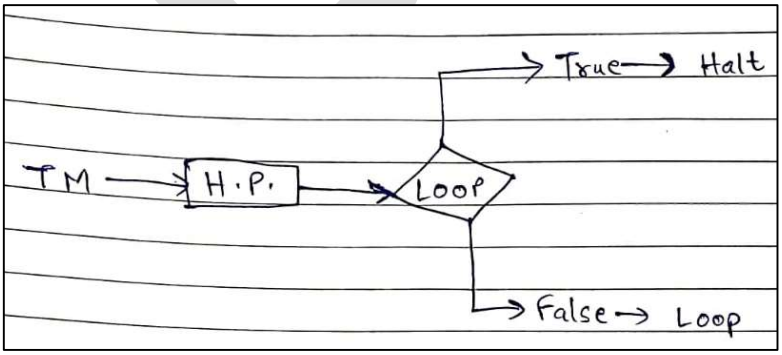
3. Distinguish between NFA & DFA

- Deterministic Finite Automata (DFA):
  - A Finite Automata (FA) is said to be deterministic if corresponding to an input symbol, there is a single resultant state i.e. there is only one transition
  - Defined as: $(Q, \Sigma, \delta, q_0, F)$
    where,
    Q – non-empty finite set of states
    Σ – non-empty finite set of input symbols
    δ – transition function which maps $Q \times \Sigma \rightarrow Q$
    q_0 – initial state
    F – non-empty set of final states
- Non-deterministic Finite Automata (NFA):
  - A Finite Automata (FA) is said to be non-deterministic if there is more than one possible transition from one state on the same input symbol
  - Defined as: $(Q, \Sigma, \delta, q_0, F)$
    where,
    Q – non-empty finite set of states
    Σ – non-empty finite set of input symbols
    δ – transition function that takes a state from Q & input symbol from Σ & returns a subset of Q
    q_0 – initial state
    F – non-empty set of final states

NFA	DFA
NFA stands for Non-deterministic Finite Automata	DFA stands for Deterministic Finite Automata
A Finite Automata (FA) is said to be non-deterministic if there is more than one possible transition from one state on the same input symbol	A Finite Automata (FA) is said to be deterministic if corresponding to an input symbol, there is a single resultant state i.e. there is only one transition
Null moves are allowed in NFA	Null moves are not allowed in DFA
NFA can change states on null moves i.e. without getting an input it can change states	As DFA does not contain null moves, state change occurs only when an input causes transition
Each pair of state & input symbol can have many possible next state	Next possible state is distinctly set
Easier to construct	Difficult to construct
Not all NFA are DFA	All DFA are NFA
NFA requires less states than DFA	DFA requires more states
Conversion of Regular expression to NFA is simpler	Conversion of Regular Expression to DFA is difficult

4. What is Halting problem? Explain with example

- The halting problem is a fundamental issue, the problem is to determine whether a computer program will halt or run forever.
- The halting problem asks whether a given algorithm will eventually halt or continue running indefinitely for a particular input.
- Halting means that the program will either accept or reject the input & then terminate, rather than going into infinite loop.



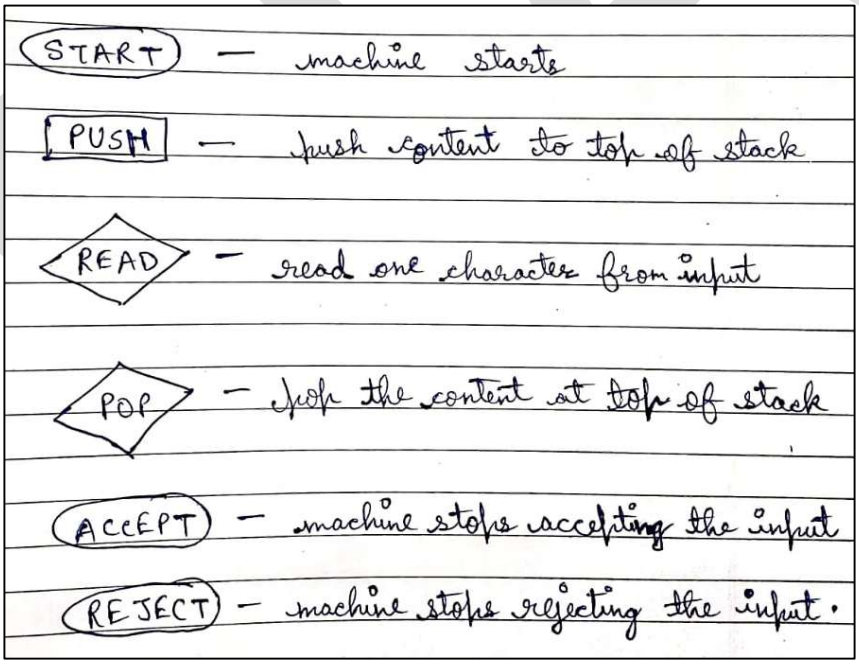
- It is impossible to design a generalized algorithm that can accurately determine whether any program will halt because the only way to know if a specific program halts is to run it & wait for outcome. Hence halting problem is undecidable problem.
- This is proved by contradiction. We provide TM & the input to the Halting problem and it predicts that if the TM halts then it will enter an infinite loop whereas if the prediction is that TM does not halt, it will stop the loop & halt immediately. Both the ideas contradict and so no general algorithm can determine whether every program will halt or not.

5. What is regular expression?

- Regular expressions are symbolic notations used to define patterns in strings.
- They describe regular languages.
- A regular expression represents a regular language if it follows these rules:
    - ε (epsilon) is a regular expression representing the language $\{\varepsilon\}$ (null string)
    - Any symbol 'a' from input alphabet Σ is a regular expression representing the language $\{a\}$.
    - Union $(a + b)$ is a regular expression if a & b are regular expressions representing the language $\{a, b\}$
    - Concatenation (ab) is a regular expression if a & b are regular expressions, representing the language $\{a, b\}$
    - Closure of 'a' i.e. (a^*) is a regular expression, meaning zero or more occurrences of 'a', forming a regular language.
- Regular expressions are used for text processing & validation.

6. Define structure of Push Down Automata (PDA). Explain power & limitations of PDA.

- Structure of PDA:



- Power of PDA:
    - Stack Memory
    - Can recognize Context-Free Languages (CFL)
    - Powerful than FA as it can handle recursive patterns.
    - PDA supports non-determinism (NPDA)
    - It needs only one stack to process input
- Limitations of PDA:
    - Limited Memory due to single stack
    - No random access to memory (as we follow LIFO in stack)
    - Not powerful enough to recognize Context-Sensitive Languages (CSL)

- Some languages that can be recognized by NPDA cannot be recognized by DPDA (like palindromes)
- Cannot handle problems requiring multi-dimensional memory

7. Define CFG

- A context-free grammar (CFG) is a formal system used to describe context-free languages (CFL)
- A grammar is said to be context-free grammar is every production is in the form of:

$$G = (V, T, P, S)$$

where,
V – set of variables (non-terminal symbols)
T – set of terminal symbols
P – set of production rules
S – start symbol (special variable from where the derivation begins)

- Non-terminal symbols (variables) represent abstract categories or placeholders.
- Terminal symbols (alphabet) are actual characters in the language.
- Production rules specify how non-terminals can be replaced with other non-terminals or terminals
- Start symbol is a special non-terminal from which derivation begins.
- Limitations of CFG:
  - CFGs are less expressive (neither English nor programming language can be expressed using CFG)
  - Can be ambiguous (can generate multiple parse trees of same input)
  - For some grammar, CFG can be less efficient because of exponential time complexity.
  - Less precise error reporting (cannot give detailed error information)

8. Describe CNF

- A context free grammar (CFG) is in Chomsky Normal Form (CNF) if all production rules satisfy the following conditions:
  - One non-terminal tends to two non-terminals
  - One non-terminal tends to a single terminal
- The idea of Chomsky Normal Form (CNF) is of a binary parse tree where each step in derivation either does not change the length of the sentential form or grows it by 1.
- A single CFG can be converted into different equivalent CNF forms.
- CNF produces same language as original CFG
- For a string of length 'n', a CNF derivation requires at most $2^n - 1$ derivation steps.
- Any CFG that does not generate ε has an equivalent CNF
- Cleaning up of grammar:
  - Remove lambda/null productions – remove nullable variables of the form $A \to \lambda$

- Remove unit productions – remove any unit productions of the form $A \rightarrow B$ which merely replace one variable by another without making any progress in derivation (causing indirection)
- Remove useless productions – remove useless variables which never lead to terminal symbols or are unreachable (does not appear on right-hand side of any production)

- Steps to convert CFG to CNF:
  - Clean the grammar
  - Replace terminals in mixed productions – eliminate terminals from RHS (if they exist) with other terminals or non-terminals (ex: $A \rightarrow xY$ can be decomposed as $X \rightarrow ZY \ \& \ Z \rightarrow x$)
  - Reduce productions with more than two non-terminals – eliminate RHS with more than two non-terminals (ex: $X \rightarrow XYZ$ can be decomposed as $X \rightarrow PZ \ \& \ P \rightarrow XY$)

9. Explain GNF

- GNF stands for Greibach Normal Form
- The idea of GNF is to make parsing linear, i.e. every step in derivation should introduce (or parse) exactly one terminal symbol from input string.

10. What is a Turing Machine (TM)? Explain the working of TM with a neat sketch. Also describe the variants of TM

- TM is defined as:

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

where,
Q – set of finite states
Σ – set of input alphabets
Γ – tape alphabet
δ – transition function
q_0 – initial state
B – blank symbol
F – set of final states

- Turing Machine (TM) can compute anything that is computable (solvable).
- It is more powerful than DFA or PDA.
- TM is a DFA with a memory unit in the form of an infinitely long tape with a linear sequence of memory cells.
- The tape has unlimited size and can solve problems of any size. It provides sequential access keeping instructions to a minimum.
- Instructions for TM:
  - Read symbol at current cell
  - Write a new symbol at current cell

    - Move left
    - Move right
- TM can produce two kinds of outputs:
    - Can compute a result from input string & write result on tape
    - If possible result of computation is given as part of input, they can give us Boolean answer by halting either in a final or non-final state of automaton indicating whether given input is true or false or whether given result is correct or not.
- Variations of TM:
    - Machine with Stay-option
    - Multi-track tape
    - Semi-infinite tape
    - Multi-tape
    - Multi-dimensional tape
    - Non-deterministic Turing machine

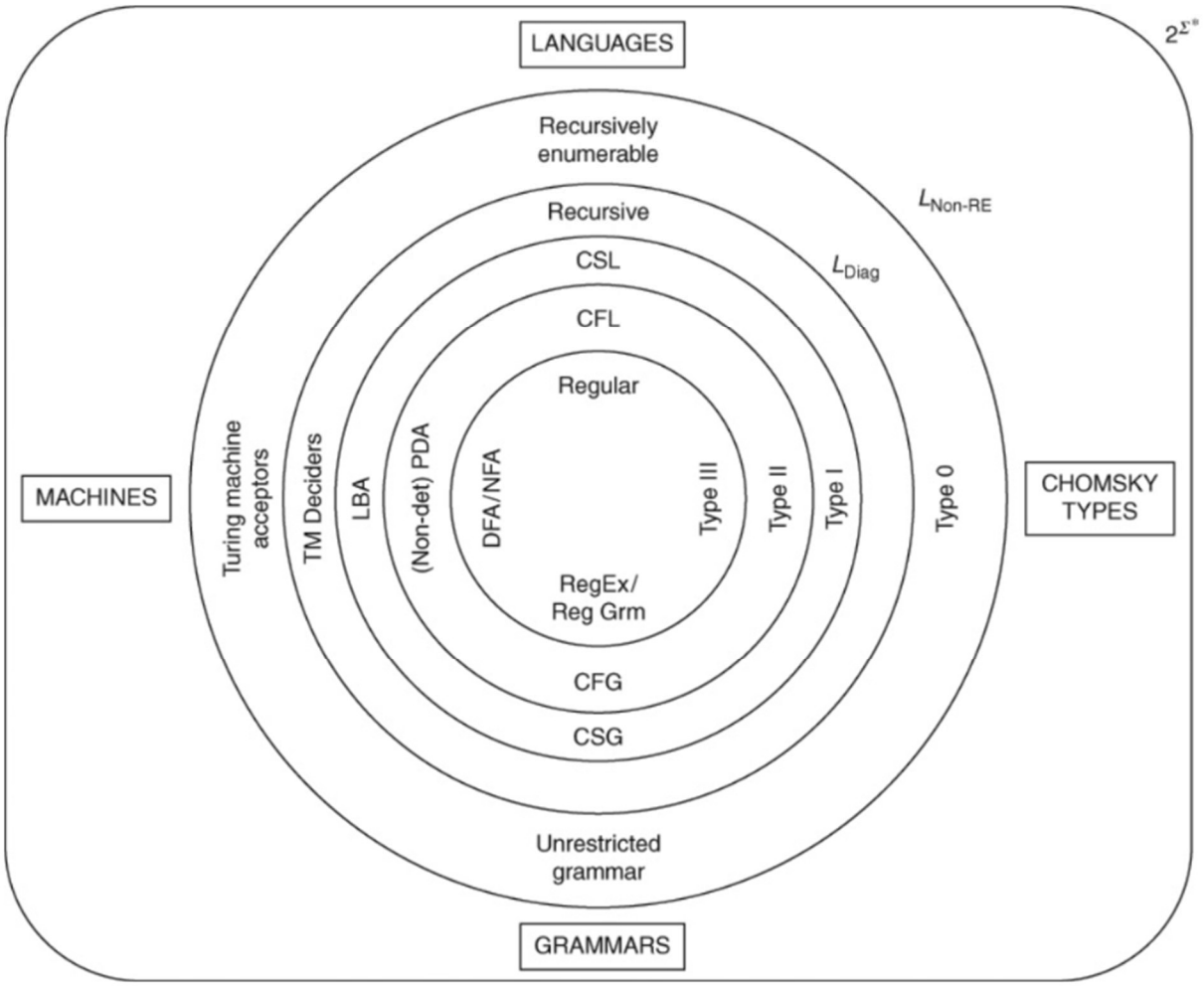
11. Applications of FA/FSM/FM, CFG, PDA & TM

- Applications of Finite Automata (FA):
    - Lexical analysis in compilers – identify keywords, operators, & tokens in source code
    - Pattern recognition with regular expressions – models regular expressions used for searching & matching patterns in text files
    - Digital Circuit Design – used in design of circuits
    - Text Editors – used to find & replace patterns in large text files
    - Spell Checkers – used to recognize patterns in large text files
    - Decision Making & Learning – can be modelled to help automate decision-making processes
- Applications of PDA:
    - Syntax Analysis in Compilers – helps parse programming language structures by using a stack to manage nested elements
    - Stack-based Applications – used in operations depending on last inserted element like arithmetic expressions
    - Tower of Hanoi Problem – solves problem involving recursive & stack-based solutions
    - Network protocols – can validate message formats & enforce structured communication
    - Natural Language Processing – used in tasks such as parsing sentences & generating syntax trees
    - Cryptography – helps in designing algorithms for encryption & decryption
    - Automatic Theorem Proving – applied to verify correctness of software models & systems

- Applications of TM:
  - Solving recursively enumerable problems – can solve any problem that is recursively enumerable
  - Artificial Intelligence – forms foundation of AI algorithms, including decision-making & machine learning
  - Robotics – used to model robot actions & control systems
  - Neural Networks – can model complex neural networks
  - Complexity Theory – used to analyze computational complexity of algorithms
  - Computational Biology – applied to model biological processes & systems
  - Quantum Computing – provides insights into relationship between classical & quantum computing models
  - Digital Circuit Design – used to model & verify behaviour of complex digital circuits
- Applications of CFG:
  - Programming Language Design & Compilers
  - Natural Language Processing (NLP)
  - XML/JSON Parsing
  - Query Languages
  - Automatic Code Generation (in IDE)

12. Draw diagram for Chomsky hierarchy & show all the types with proper explanation.

- Diagram:



- Table:

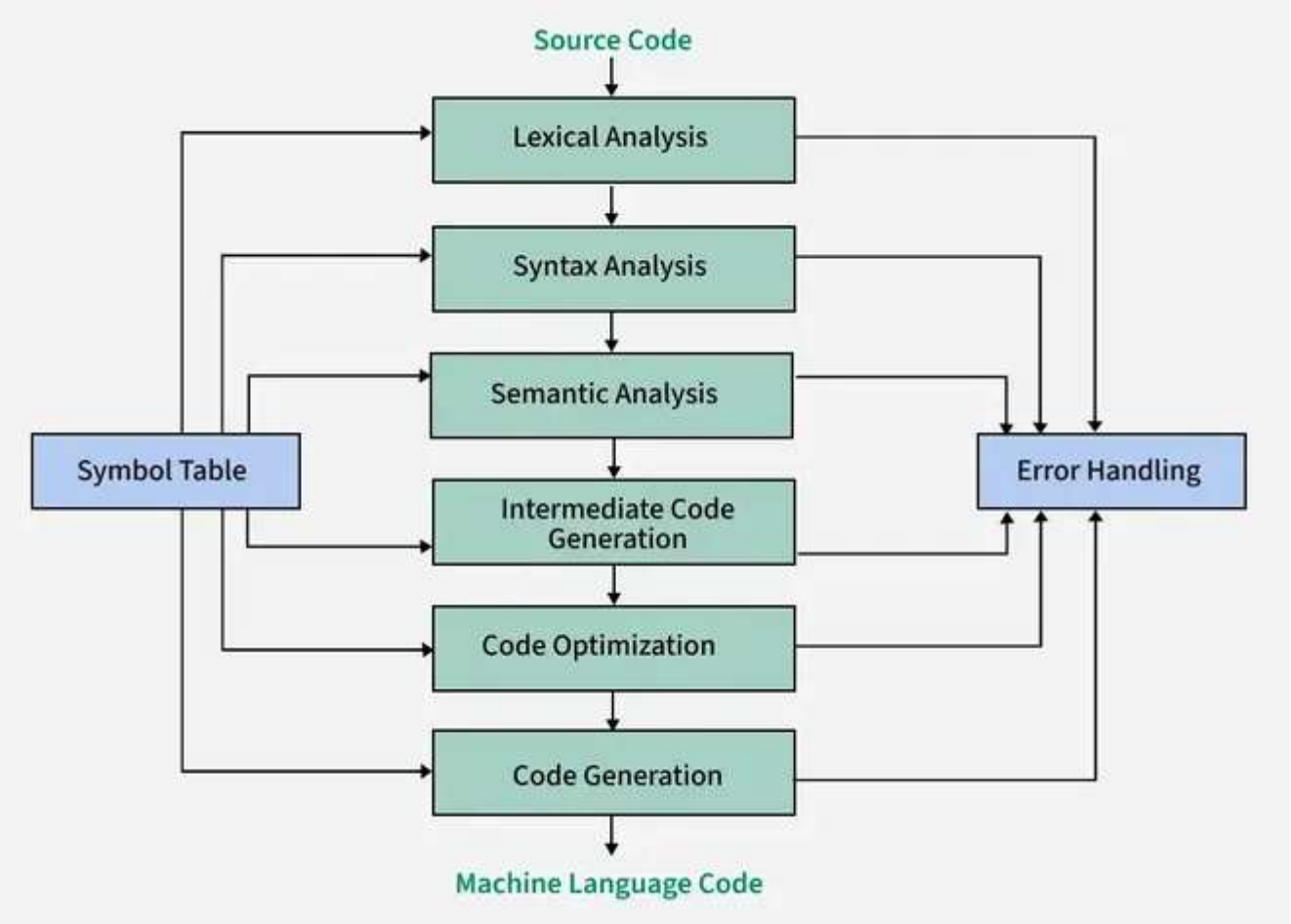
Type	Grammar	Language	Machine
Type 0	Unrestricted grammar	Recursively enumerable	Turing machine
Type 1	Context Sensitive Grammar (CSG)	Context Sensitive Language (CSL)	Linear Bounded Automata (LBA)
Type 2	Context Free Grammar (CFG)	Context Free Language (CFL)	Push-Down Automata (PDA)
Type 3	Regular Grammar	Regular Languages	DFA/NFA

13. Right Linear & Left Linear Grammars

- Linear grammar refers to a grammar having only one non-terminal on the right-hand side of every production. A regular grammar is a right-linear or left-linear grammar.
- Right-linear grammar is grammar where non-terminal is the rightmost symbol on right-hand side of every production.
- Left-linear grammar is grammar where non-terminal is the leftmost symbol on the right-hand side of every production.
- Some productions being right-linear & some others being left-linear in the same grammar is not allowed
- For every right-linear grammar, there is an equivalent left-linear grammar whose language is the same as that of the right-linear grammar.
- For every right-linear grammar, there is an equivalent NFA that accepts the same language as the language of the grammar.

14. What is a compiler? Describe different phases of a compiler

- A compiler is a tool that converts high-level programming code into machine code that a computer can understand & execute. It acts as a bridge between human-readable code & machine-level instructions, enabling efficient program execution.



- Phases of a Compiler:
  - Lexical Analysis
    - The first phase of compiler where source code is scanned character by character & converted into a sequence of tokens (smallest meaningful units of programming language).
    - These tokens (include keywords, identifiers, constants, operators & punctuations) are identified based on language's syntax rules.
    - This process simplifies code for subsequent stages of compilation.
  - Syntax Analysis
    - The second phase of compiler that checks whether sequence of tokens from lexical analysis follows grammatical rules of programming language.
    - It verifies correct code structure, such as proper used of operators & parentheses & generates errors if syntax is invalid.
    - This phase uses parse trees & syntax trees to represent code's structure. (Parse trees show detailed grammar rule applications while syntax trees provide a simplified, abstract view of code's hierarchy)
  - Semantic Analysis

        - This checks logical correctness of source code ensuring it adheres to programming language's rules beyond syntax.
        - It detects semantic errors such as type mismatches & undeclared variables by validating operations & data usage.
    - Intermediate Code Generation
        - Intermediate code is machine-independent representation of a program generated between source code & final machine code in compilation process.
        - It serves as a bridge that enhances portability & simplifies optimization. Because it is not tied to specific hardware, it supports platform independence & enables easier implementation of optimizations like dead code elimination, loop optimization, & common subexpression elimination.
        - Its structured form makes final translation to machine code more efficient.
    - Code Optimization
        - The process of improving intermediate or target code to enhance performance making the program run faster, use less memory or operate more efficiently without changing its functionality.
        - Includes both machine-independent & machine-dependent techniques.
        - Common methods include constant folding, dead code elimination, loop optimization, & strength reduction.
    - Code Generation
        - It is final phase of compiler where intermediate representation of program is translated into machine or assembly code specific to target platform.
        - This output, known as object code, must accurately preserve the meaning of original source code & be efficient in terms of CPU & memory usage.

15. Reduced DFA

16. Define following terms & give example of each: Automata, String, Language, Alphabet, Grammar

- Automata – the term 'automata' means 'self-acting'. An automaton is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.
- String – A string is a finite-sequence of symbols taken from Σ (ex: abcdbcd is valid string on alphabet set $\Sigma = \{a, b, c, d\}$
- Language – A language is a subset of Σ^* for some alphabet Σ. It can be finite or infinite (ex: If language takes all possible strings of length 2 over $\Sigma = \{a, b\}$, then $L = \{ab, aa, ba, bb\}$)

- Alphabet – An alphabet is any finite set of symbols (ex: $\Sigma = \{a, b, c, d\}$ is alphabet set where a, b, c, d are symbols)
- Grammar – Grammars are formal systems used to define structure of languages by specifying how strings can be generated. They provide a set of rules for creating valid sentences or expressions within a language. (ex: $(a + b)^*$)

17. What are limitations of FA?

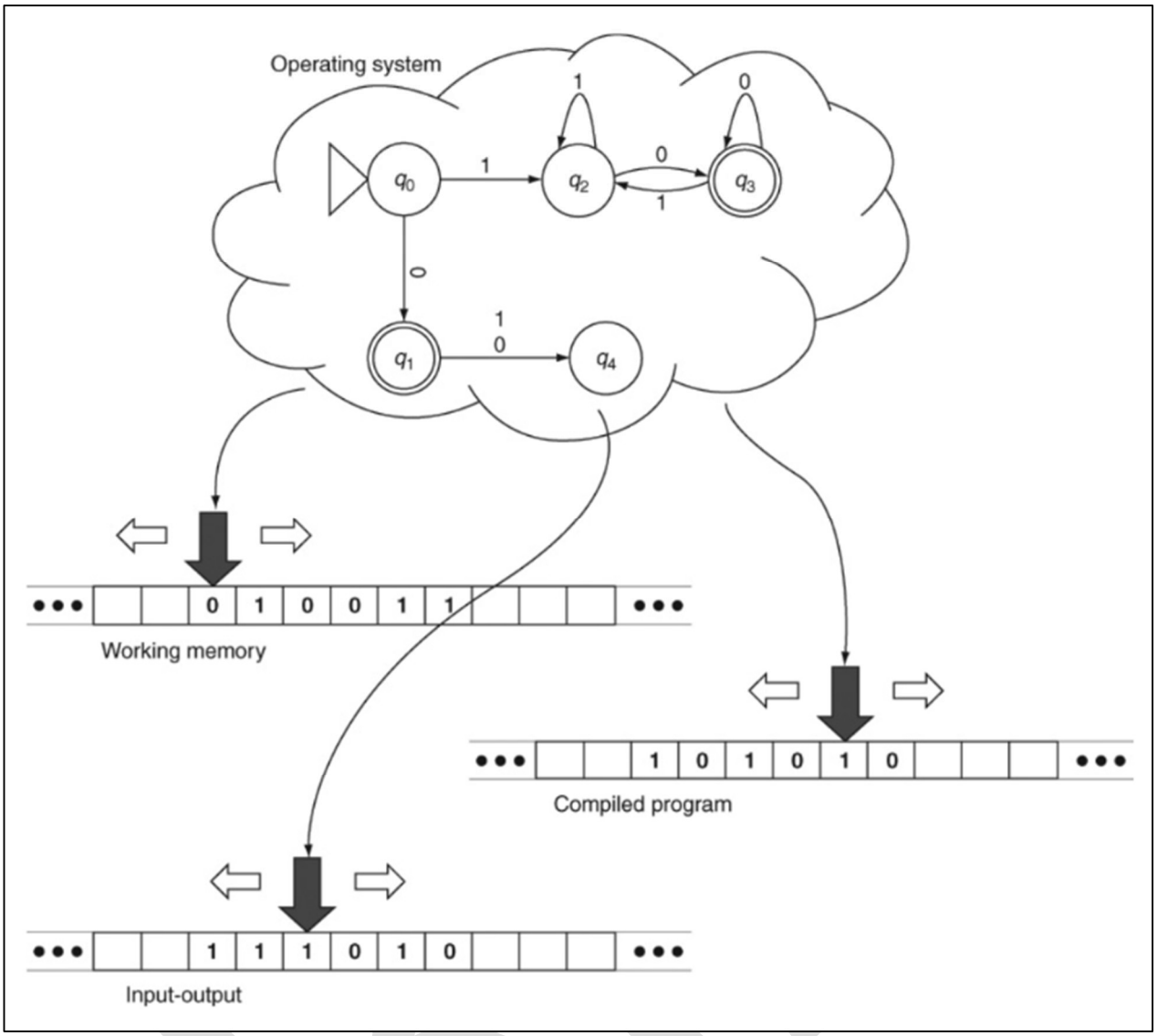
- FA contains no memory (no memory beyond states)
- FA cannot count number of occurrences
- FA can only recognize regular languages so it does not support non-contextual processing
- FA cannot handle nested structures (like $a^n b^n$)
- Finite automata cannot do backtracking
- FA does not hold computational power for sorting, swapping, etc.

18. What is DFA?

- Deterministic Finite Automata (DFA):
    - A Finite Automata (FA) is said to be deterministic if corresponding to an input symbol, there is a single resultant state i.e. there is only one transition
    - Defined as: $(Q, \Sigma, \delta, q_0, F)$
      where,
      Q – non-empty finite set of states
      Σ – non-empty finite set of input symbols
      δ – transition function which maps $Q \times \Sigma \to Q$
      q_0 – initial state
      F – non-empty set of final states
- DFA with output is in two variations i.e. Moore machine & Mealy machine
- DFA gives answers in the form of Boolean (True/False or Yes/No or Accept/Reject)

19. Universal Turing machine

- A Universal Turing machine is a “programmable” machine that can compute anything that is computable.
- A Universal Turing machine should be able to simulate the computation carried out by any hardcoded Turing machine
- It has three tapes:
    - Input-output tape: It is used to simulate the tape of the particular machine which it is simulating. The input & output is written on this tape.
    - Compiled program tape: It stores “the program” i.e. the table of instructions or an encoding of the transition functions of the Turing machine which it is simulating. This tape is a read-only tape.
    - Working memory tape: The tape on which the current state of the machine being simulated is maintained, along with any working memory it needs.



20. Discuss power & limitations of PDA. Compare it with FA & TM

- Power of PDA:
  - Stack Memory
  - Can recognize Context-Free Languages (CFL)
  - Powerful than FA as it can handle recursive patterns.
  - PDA supports non-determinism (NPDA)
  - It needs only one stack to process input
- Limitations of PDA:
  - Limited Memory due to single stack
  - No random access to memory (as we follow LIFO in stack)
  - Not powerful enough to recognize Context-Sensitive Languages (CSL)
  - Some languages that can be recognized by NPDA cannot be recognized by DPDA (like palindromes)
  - Cannot handle problems requiring multi-dimensional memory

- Comparison of PDA, FA & TM:

Feature	PDA	FA	TM
Languages recognized	CFL	Regular languages	Recursively enumerable languages
Memory	Stack	No memory	Infinite tape
Arbitrary problems	Cannot solve	Cannot solve	Solvable if computable
Turing Complete	No	No	Yes
Nested structures	Supported	Not supported	Supported
Backtracking	Limited	Not supported	Supported
Power	Medium	Low	High