Professional Online Academy Vu Topper RM

CS402-Theory Of Automata
Update Important Subjective
Mid Term by Vu Topper RM

Follow Us on YouTube:
@vutopperrm

For More Help Contact What's app Me!!!
Number's = # 03224021365, 03164980064

Rizwan Manzoor

بری صحبت سے تنہائی بہتر ہے اور تنہائی سے نیک صحبت بہتر ہے

For More Help Contact What's app 03224021365

Very Important Topics with Question.

1) What is FA.

Sol: In theory of automata, FA is a Finite Automaton, a simple machine that recognizes patterns in a sequence of symbols. It has a limited number of states and transitions based on input symbols. It accepts the input if it ends in a designated "final" state.

2) What is NFA.

Sol: In theory of automata, NFA stands for Non-Deterministic Finite Automaton. It's a type of FA where:
It can have multiple possible transitions from a single state for the same input symbol. (Think: choosing a path)
It can sometimes move between states without any input (using epsilon transitions). (Think: taking a shortcut)
This makes NFAs more flexible for representing certain patterns than regular FAs.

3) What is GTG.

Sol: In theory of automata, GTG stands for Generalized Transition Graph. It's a way to represent a machine that's more powerful than a Finite Automaton (FA).

4) Kleene star closure.

Sol: In theory of automata, the Kleene star (Σ*), applied to an alphabet (Σ), represents all possible strings of any length over that

alphabet, including the empty string (λ). It basically means "zero or more repetitions" of symbols in the alphabet.

5) Convert NFA into FA.

Sol: NFAs can be converted to Deterministic Finite Automata (DFAs). This process involves finding the epsilon-closure (all reachable states with epsilon moves) for each state in the NFA and building the DFA based on those closures. While NFAs can be simpler to design, DFAs are generally more efficient for simulation.

6) What is the transition table.

Sol: A transition table is a table in automata theory that shows what state a machine (like a finite-state automaton) will move to based on its current state and the input it receives. Rows represent current states, columns represent input symbols, and entries show the next state. It's a compact way to represent the behavior of the automaton.

7) Convert given TG into FA.

Sol: Unfortunately, I can't directly convert a Transition Graph (TG) into a Finite Automaton (FA) description (like a transition table) without the actual TG. However, I can give you a short answer on the conversion process:

Identify States: Each node in the TG represents a state in the FA. Identify Start and Accepting States: Mark the designated start and accepting states in the TG as the same in the FA.

Analyze Transitions: For each transition arrow in the TG labeled with an input symbol (a from the alphabet), create a corresponding entry in the FA's transition table. The row represents the source

state (from the arrow's tail), the column represents the input symbol, and the entry shows the destination state (at the arrow's head).

8) What is a regular expression.

Sol: In theory of automata, a regular expression is a concise way to describe a set of strings (a language) recognized by a finite automaton (FA). It uses symbols and operators to define patterns that strings must follow to be considered part of the language.
Regular expressions are powerful for representing simple patterns like:
Specific characters (a, b, etc.)
Any character in the alphabet (.)
Repetition (a*, meaning zero or more "a"s)
Choice (a|b, meaning either "a" or "b")

9) Length and reverse of string.

Sol: Finite Automata (FA) are not powerful enough to directly determine the length or reverse a string. They are limited to recognizing patterns within a string based on state transitions.
Length: FAs can't count the number of symbols processed during transitions. They only accept or reject based on the final state reached after reading the entire string.
Reverse: FAs operate by reading the input symbol by symbol, and their transitions depend on the current state and the current symbol. They don't inherently can process the string in reverse order.

10) String was given to make CFG.

Sol: Absolutely! We can attempt to create a Context-Free Grammar (CFG) from a given string. Here's a basic approach:

Identify Terminal Symbols: These are the actual characters that appear in the string (a, b, etc.).

Break Down the String: Try to split the string into smaller sub-strings that can be combined using rules. There might be multiple possible breakdowns depending on the string's complexity.

Introduce Non-Terminal Symbols: Represent the sub-strings with non-terminal symbols (X, Y, etc.).

Define Production Rules: Create rules that show how non-terminal symbols can be rewritten as terminal symbols or combinations of other non-terminals. These rules capture the structure of how the string is built.

11) Summary table of row language.

Sol: In theory of automata, there isn't a concept of "row language" associated with the tables used for finite automata (FA).

Here's a breakdown to clarify:

Tables in Finite Automata:

These tables represent the behavior of an FA.

Rows correspond to the states of the automaton.

Columns correspond to the input symbols the automaton can process.

Entries in the table represent the next state the automaton transitions to when it reads a particular symbol in a specific state.

12) Moore mealy machine difference.

Sol: Moore vs. Mealy Machines

Both Moore and Mealy machines are finite state machines that can generate outputs. The key difference lies in how their output is determined:

Moore Machine: Output depends only on the current state.
Simpler design, but output might be one step behind input changes.
Mealy Machine: Output depends on both the current state and the current input. Reacts faster to changes but can be more complex.

13) Question about ambiguity of CFG.

Sol: A Context-Free Grammar (CFG) is ambiguous if a string in the language can have more than one parse tree.
In other words, the grammar allows for the same string to be derived in multiple ways.

14) Moore Machine & Mealy Machine.

Sol: Absolutely, both Moore and Mealy machines are fundamental concepts in automata theory, dealing with finite state machines (FSMs) that generate outputs based on inputs and their current state. Here's a quick breakdown of their key differences:

Moore Machine:

Output depends solely on the current state.
Requires more states due to separate output logic.
Slower output reaction (one clock cycle delay).
Simpler design.

Mealy Machine:

Output depends on both the current state and the current input.
Needs fewer states as output is tied to transitions.
Faster output reaction (same clock cycle).
More complex design due to combined input-output logic.

15) Transition graphs define with example.

Sol: Transition Graphs: In automata theory, a transition graph (TG) is a visual representation of a finite automaton (FA). It depicts the behavior of the FA as it processes an input string. Here's a breakdown of its components:

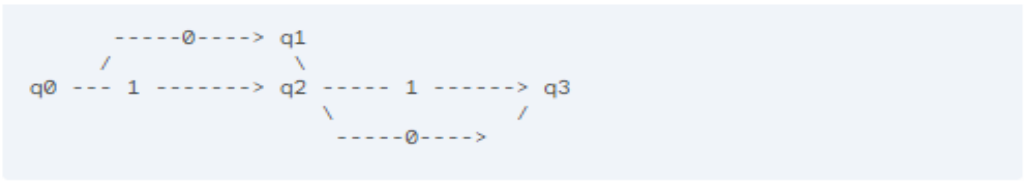
States (Q): A finite set of circles representing the possible configurations of the FA. At least one state is designated as the start state (often marked with an incoming arrow), and some states are designated as final states (often marked with double circles).

Alphabet (Σ): A finite set of symbols that the FA can read as input.

Transitions: Directed arrows connecting states, labeled with input symbols from the alphabet. A transition from state q to state p labeled with a indicates that the FA moves from state q to state p when it reads the symbol a.

Example:

Consider a finite automaton that recognizes strings starting with "01" and ending with "10." Here's its transition graph:



- **States:** `q0` (start), `q1`, `q2`, `q3` (final)
- **Alphabet:** {0, 1}
- **Transitions:**
  - q0 -> q1 (reading 0)
  - q0 -> q2 (reading 1)
  - q1 -> q2 (reading 1)
  - q2 -> q3 (reading 0)
  - q3 -> q0 (reading 1) (This loop ensures the string ends with "10")

This FA accepts strings like "0110", "010110", etc., but rejects strings that don't begin with "01" or end with "10".

16) r1=a*+b* and r2=(a+b)* in differentiate.

Sol: r1 and r2 differ in their ability to accept strings with both a and b. r1 can only handle strings with all as or all bs, while r2 can accept any combination of as and bs.

17) Given sigma start contains odd a's or not.

Sol: An automaton for strings starting with sigma (Σ) wouldn't directly tell you about odd "a"s. It focuses on the starting symbol, not counting occurrences.

18) Difference B/W Distinguishable and indistinguishable.

Sol: In automata theory:

Distinguishable strings (w.r.t. a language L): Two strings are distinguishable if there exists a string you can add to them such that adding it makes them land in different categories (belong to or not belong to the language L).

Indistinguishable strings (w.r.t. a language L): Two strings are indistinguishable if no matter what string you add to them, they always end up in the same category (both belong to L or both don't belong to L).

Think of them like paths in an automaton. If adding a symbol can lead them to different states (accepted/rejected), they are

distinguishable. If they always end up in the same state, they are indistinguishable.

19) Recursive define the factorial defined over sigma=1,2,3,

Sol: Finite state automata (FSA) are not ideal for defining recursive functions like factorial. They are better suited for recognizing patterns in strings.

However, we can describe a recursive definition of factorial similar to how it's done mathematically:

Base Case: "1" is a valid representation of factorial (1! = 1).

Recursive Case: If a string ends in "n" (where n is any of the symbols 1, 2, or 3), then "n!(n-1)" is also a valid representation of factorial (n! = n * (n-1)!).

This definition captures the essence of recursion: a smaller instance of the problem (n-1)! is used to define a larger one (n!).

But an FSA wouldn't be the best tool to implement this due to its limitations in handling recursion.

20) The language can express in FA then why we need NFA. Justify your answer.

Sol: Even though both Deterministic Finite Automata (DFA) and Non-Deterministic Finite Automata (NFA) can recognize regular languages, NFAs offer some advantages:

Simplicity: Often, an NFA can be constructed much simpler than a DFA for the same regular language. NFAs can leverage "epsilon transitions" (ε-moves) to explore multiple possibilities at once, making them more concise for specific patterns.

Power equivalence: Despite the simpler design, any language recognized by an NFA can also be recognized by a DFA (though the DFA might be more complex). This ensures we don't lose expressive power by using NFAs.

In essence, NFAs offer a more compact way to represent certain regular languages, even though DFAs are ultimately just as capable.

21) Find Regular Expression and draw GTG Diagram.

22) Reverse of a string.

23) Draw the diagram and regular expression in GTG that's not starting with a, but ending with a.

24) Draw the diagram and regular expression in TG that's Containing triple aaa and Triple bbb.

25) What is regular language? Tell and draw the diagram that's ending and starting with same letters.

26) The language, defined over Σ ={a, b} of words containing double aa is fa create in regular expression.