**CCD**

**🔹DFA (Deterministic Finite Automaton)**

**Definition**:

A **DFA** is a 5-tuple machine defined as:

**DFA = (Q, Σ, δ, q₀, F)** where:

* **Q** = Finite set of states
* **Σ** = Finite set of input symbols (alphabet)
* **δ** = Transition function: **δ: Q × Σ → Q**
* **q₀** ∈ Q = Start state
* **F** ⊆ Q = Set of accepting (final) states

**Key Properties**:

* For each state and input symbol, **only one transition** is defined.
* No ε (epsilon) transitions are allowed.
* Machine can be in **only one state at a time**.

**🔹NFA (Nondeterministic Finite Automaton)**

**Definition**:

An **NFA** is a 5-tuple machine defined as:

**NFA = (Q, Σ, δ, q₀, F)** where:

* **Q** = Finite set of states
* **Σ** = Finite set of input symbols (alphabet)
* **δ** = Transition function: **δ: Q × (Σ ∪ {ε}) → 2^Q**
* **q₀** ∈ Q = Start state
* **F** ⊆ Q = Set of accepting (final) states

**Key Properties**:

* For a state and input symbol, **multiple possible transitions** may exist.
* **Epsilon (ε)** transitions are allowed (transitions without consuming input).
* Machine can be in **multiple states at the same time** (nondeterminism).

**🔄 DFA vs NFA (Quick Comparison)**

| **Feature** | **DFA** | **NFA** |
| --- | --- | --- |
| Transition | One per input | Zero, one, or many |
| Epsilon (ε) transitions | Not allowed | Allowed |
| Determinism | Deterministic | Nondeterministic |
| Complexity to simulate | Easy | Harder (may need conversion) |
| Expressive power | Same as NFA | Same as DFA |

Summary

**Automata Fundamentals**

* Automata is an abstract machine model used for language representation
* Used to determine whether a language is acceptable or not
* DFA (Deterministic Finite Automata) means applying an alphabet to a state takes you to exactly one particular state
* NFAs (Nondeterministic Finite Automata) allow multiple choices/transitions from a state

**Formal Definition of DFA**

A DFA is represented by 5 components:

* Q: Set of all finite states
* Σ (Summation): Set of alphabets (e.g., A and B)
* δ (Delta): Transition function showing state changes
* q₀: Start state (initial state)
* F: Set of final/accepting states

**DFA Design Example**

Example: Design a DFA for language of strings starting with 'A'

* Start with the minimum requirement (at least one 'A')
* Create q₀ as start state
* Create a final state q₁ that is reached when 'A' is read from q₀
* Create a "trap state" q₂ for 'B' from the start state
* Make self-loops for both A and B in q₁ (as any string can follow the initial A)
* Make self-loops for both A and B in q₂ (trap state)

**Transition Representation**

* Transitions can be represented in a table format
* Cross-multiply states (Q₀, Q₁, Q₂) with alphabet symbols (A, B)
* For each state-symbol pair, specify the next state
* Ensures all possible transitions are defined for a complete DFA

Notes

Transcript

It can go in any state. There are multiple choices in the case of NFA. Deterministic finite automata means that if you apply any alphabet in a state, it will take you to a particular state. This is the funda of deterministic. What is automata? We have already discussed it.

Automata is a machine that uses language representation. If we want to represent language in a standard way, whether my language is finite or infinite, if I want to represent it, Automata is an abstract model that tells us whether a language is acceptable or not. This is called Deterministic Finite Automata. How do we denote DFA? We denote DFA with these 5 symbols. First is Q. Q stands for set of all finite states.

Set of finite states, means what all states are there in your machine. What is the meaning of states? One state is the starting state. To go to the output, how many transitions did you do in between? How many states did you go to? So we call the collection of all those states as a cube.

Then we have summation here. What is this in action? This is what I have already discussed. This is the alphabet. Set of alphabet. Let's suppose we take the formula. For example, we take A and B. That is, where is A going? Where is B going? We also take 0 and 1.

But most of the time we use A and B alphabet. Then we have delta. What is this in action? Transition. Transition means that you have applied any alphabet on a state, A or B. Where did you go? I mean, where did you reach? What is the meaning of that? If you reach a state, then this transition is actually called, Transition from one state to another state is actually denoted by a degree.

Then Q note. Q note is your start. Which state did you start from? F is the set of finite final states. Generally, we have only one final state. The final state is where your output will be generated. But sometimes, we have multiple final states. So, we call it the set of all final states.

But remember, how many total states you have? DFA is one of the most important topic of TOC because maximum question you get in competitive exams is related to DFA What is DFA? Why do we use it? How do we use it? and then design it, which we will discuss in the next videos So here is the basic definition of these 5 points that how do we denote DFA So let's take a simple example Let's suppose I want to make a DFA I want to make a deterministic finite automata for a language I want to make a DFA I want to make a deterministic finite automata for a language I want to make a DFA

It has all the strings.

But can it be B or A? No. You have to remember that the starting letter is B, which means there should be no strings of this type. So what does this actually mean? Start with A. So this is obviously finite or infinite. This is the part of the infinite language.

Now I have to represent this infinite language with the help of finite automata. How will we do it? Let's start with the designing part. So let's design. First of all, we have Qnote. That is our initial state. So let's denote the starting state or initial state like this.

Qnote. I mean, it can also be multiple. In some places, it can be denoted as A, B, C. X, Y, Z. So there is no need to worry. This arrow indicates the start state. I want A. Always remember, when designing DFA, first design for minimum to minimum string. Then fill in the blanks. Fill in the blanks means, what is minimum to minimum? I want at least one A.

What does it mean? Is it a DFA? No. Why is it not a DFA? What is the reason for this? It is not a DFA because in every DFA, if I talk about alphabets, let's suppose that in this question, the alphabets we have are A and B. So see, if it is for A, it will go here.

But where is it going for B? We have not even denoted it for B. So I will have to denote for all the alphabets. So why is it going here for A? Where will it go for B? If I make B here, let's suppose if I make B here, then my question is wrong. Because if B comes in the starting, then B will also get accepted.

So that means I will make B here. Let's suppose I have made a self-loop for B. If I make a self-loop for B, and the string is BA, how do I run it? B comes from the starting, we stay on the Q note, and when A comes, we reach here. This is accepted. But we don't have to accept BA. A should come in the starting.

This means that a self-loop cannot be made, and neither can B be written here. So obviously, I have to trap B. I have to put B in a dead state, so that I can't come back from there. This is called a trap. We have made a state for B, let's say Q0. And whenever B comes, it will go to Q0. It can't go anywhere from Q0. That means there is no path to reach to the final state. We have trapped it. We have shifted it to a dead state. So we have made it in a dead state.

Now look, Q0 is going here to A and B is going here. But Q1, where is Q1 going? We have neither made Q1 on A nor B. So there is no problem. Now make it simple on A and B. How did it come? How did it come to your mind that you have to make A and B in a self loop?

Look, the starting letter is A. So that we can keep it as a trap. It can't come out. It can't come up. Nor can it go anywhere else. So this is your DFA. Because Q0 is going to A and B. Q2 is going to A and B. This is also fine. Q1 is going to A and B. This is also fine. So now your DFA is ready.

So this is called transition. What does transition mean? We denote transition in this way. Transition means Q into... That is A and B, and it is 1. Representing the Q. We represent delta in this way. What are the states of Q? Q0, Q1, Q2. Multiply it by Q0, Q1, Q2. Multiply it by summation. What are the alphabets in this question? A and B. If you multiply these two, you will get Q0A, Q0B, Q1A, Q1B, Q1B.

Q2A, Q2B. We have simply cross-multiplied both of them. So, we have 3 into 2 is 6 states. So, we have to name these 6 states as Q. What is Q here? Q0, Q1, Q2. So, see how you will represent. Q0 is A. If A is on Q0 and Q0, then where will you go? You will go to Q1.

So, you denote it like this. So, this is your transition. If B is on Q0 and B is on Q0, then you will go to Q2. So, make it Q2. Then, if A is on Q1, then you will be on Q1. If A is on Q1, then where will you reach? You will not reach anywhere. You will be on Q1.

If B comes on Q1, where will you go? You will stay on Q1. Similarly, if A comes on Q2, you will stay on Q2. If B comes on Q2, you will stay on Q2. So, this is the transition to represent the funda. So, Q1, your Q0, Q1 and Q2 are completed like this. So, what is this transition?

If I apply alpha-bet on a state, where will you go? That is called a transition. And this is a basic example starting with A, Deterministic Finite Automata. So this is all important points related to the DLQ. Thank you.

Summary

**Introduction to DFA Construction**

This lecture explains how to construct Deterministic Finite Automata (DFAs) for two common language patterns: strings containing 'A' and strings ending with 'A'. These examples are relevant for competitive exams and university-level computer science courses. The lecture assumes basic knowledge of DFAs, with a reference to an introductory video available in the description.

**DFA for "Strings Containing A"**

* **Language definition**: All strings from the alphabet {A, B} that have at least one 'A'
* **Minimum acceptable string**: "A"
* **Construction approach**:
  + Create a starting state (q0) and an accepting state (q1)
  + From q0, transition to q1 when 'A' is read
  + From q0, stay in q0 when 'B' is read (no 'A' seen yet)
  + From q1, stay in q1 for both 'A' and 'B' inputs (once an 'A' is seen, all subsequent strings contain 'A')
* **Key insight**: After seeing an 'A', all longer strings will also contain 'A', so they should be accepted

**DFA for "Strings Ending with A"**

* **Language definition**: All strings from the alphabet {A, B} where the last symbol is 'A'
* **Minimum acceptable string**: "A"
* **Construction approach**:
  + Create a starting state (q0) and an accepting state (q1)
  + From q0, transition to q1 when 'A' is read
  + From q0, stay in q0 when 'B' is read
  + From q1, stay in q1 when 'A' is read (still ends with 'A')
  + From q1, transition back to q0 when 'B' is read (no longer ends with 'A')
* **Key insight**: The accepting state represents "just saw an A" and this state is only maintained if another 'A' follows

Notes

Transcript

Today I am going to explain how to construct DFA And I will discuss two examples related to this in this video Which is very beneficial for your competitive exams, even for your college and university level exams So guys, quickly like the video, subscribe to the channel if you haven't done it yet And please press the bell button so that you get all the latest notifications So let's start Construct a DFA which asserts a language of all strings containing A My first question is containing A And the second question we will discuss is end with A First of all, let me tell you what is DFA?

I have already made a video on this and the link is given in the description box. So do check that video once so that you get to know the introduction to DFA. What is actually DFA? Because I will not explain that point again and again in this. So here we directly start containing A.

What does containing A mean? All the strings in the language should be such that A should come inside it. So you always start with minimum. What does minimum mean? Which will be the minimum string from minimum? Which will be the smallest string? A itself.

Your smallest string will be A. What else can happen? Let's suppose we have A, A, B, C. What do I have in the starting state?

Now, filling double x. Filling double x means that you have made it for single a. Now, how to complete the combination of multiple a, b, a, a, b? For that, what you have to do is, in BFA, filling double x. Filling double x means, where did q0 go on a?

It went here. And the summation that we have, let me tell you the alphabets, there are only two alphabets, a and b. Because if you look at the question carefully, sometimes a, b, c are given, extra is also given. So, here I have only two symbols, a and b.

So, it went here for a. But where will q0 go for b? Where should it go for b? For b, we can either make it here. This is one possibility. What is the second possibility? I will say that it is going here for B. This is also a possibility. What is the third possibility? I will put it in a tap state. So first of all, why should we take it in a tap state? Because if B comes to A, then what is my question?

Is it containing A? So A is coming inside B. But if you put B in a tap state, then it will tap. It will never accept B. Like we explained starting with A. In starting with A, if you make it like this, then B will go to A. If it is on Q2, then it will never be accepted. But it should be accepted. So to make it accepted, obviously I don't have to create a trap state. This is how you have to apply the logic. Second, if I run B like this, that it is going here for A and here for B.

If it is going here for B, then B is accepted alone. A single B is accepted. Then my question will be wrong containing A. So that means I can't make this B. So the third possibility is, put B in the self loop. So put B in the self loop and accept this.

So look, BA will come, it will be accepted. B, BA will come. B, A, A. Next A comes, it goes into the trap state. Why does it go into the trap state? B, A, A. A is also coming, so containing A, A should be accepted. So, whatever A comes, whatever A comes, what does it mean? You put A in the trap state, you make A self-true.

That is, whatever A comes, at least we didn't give A for A. Apart from that, whatever A comes, that should also be accepted. Yes, that will also be accepted. So, where will it go for B? For B, it can go into the trap state. For B, it can go into the trap state.

If I send B to the trap state, then what will happen? Let's suppose I have a string, A, A. Be cool.

A, B. A, A, B. So, it didn't get accepted. It should have reached here on B. But, it will reach here. Because, it is stopped on Q-merge. So, this string is mine. It didn't get accepted. But, it should get accepted. So, what do you have to do to get it accepted?

Instead of making it here, make a selector for A and B. That's it. As many A's as you want. This is your simple DFA, which is containing A.

I always had a burning desire to push myself further. Hello friends, welcome to Gate Smashers. Next example is construct a DFA which accept a language of all strings end with A. In last video, we have discussed about containing A. In this video, we are discussing how to construct a DFA which will accept all the strings which must end with A.

So, what is your first step? Your first step is to write a language. And what is a language? The language is a collection of string. So here, in end with a, which is the minimum string that I will get? Although, the minimum string that we have is epsilon. But look, in epsilon, a doesn't come anywhere. That means, we have only written what should be the end with a.

So, what will be the minimum string that you will get? a. Apart from this, obviously, keep moving it forward. a, a, a, a, a, like this. Apart from this, what else can come? Can b, a come? Yes. Look, in b, a, a is coming last. So, yes, b, a, b, a, b, a can come like this.

a, b, a. Because in this, B and A are coming at the end. Can A and B come? A, B, A, B, A, B. Can strings come like this? No. This cannot come. Because what is coming at the end? B is coming. And end with A should be there. So this string should not come inside this.

So first of all, you write the language. Which strings should come inside this? So this is the language that we have. And what is this? It is the infinite language that is being made here. So now we will design its DFA. In end with A, when we design the DFA, you have to design for minimum to minimum strings.

So what I have is, first of all, we have the initial state or starting state of Q0. And on Q0,

A and B. So where is Q0 going to A? It is going to Q1. And Q1 is my final state. And look, A, I mean, single A came. So what was your minimum string? Single A. So what did we do with single A? We sent it to the final state. And here, if I make a loop of this.

If I make a loop of A here, then not only single A, but multiple A can also come. But first of all, I have to set it for minimum string. So minimum string is single A. So if you make a loop, then it is not single A. A can come as much as you want. So that means, if you apply the symbol A from Q0, then you will reach Q1.

And what is in Q1? Now what you have to do is fill in the blanks. What does fill in the blanks mean? BFA has not yet been completed. Because the cue note is going here on A. Where is it going on B? You have to make it on B also. So what is the logic to make it on B? All the strings we have, A should come at the end.

So can B come before A? Yes. B can come before A. Because we have already discussed that B, A, B, A strings can come. So should A not come before A? No. All the B's that can come before A, A should come at the end. So what do you do? Make a self loop of B.

Why did you make a self loop? Self loop is to make a self loop of B. Self loop is to make a self loop of B. Simple logic is that at the end, A should come, so before A, it can come in 1, 2, 1000, whatever it is. So you make it a self loop. We don't have to put it in any trap state or end state. Make it a simple loop. So this is going here for B and the transition is here for A.

Look at Q1. We haven't made a transition anywhere for A and B. So where should Q1 go on A? According to you, where should it go on A? Simple logic is, end with A. What does end with A mean? If my symbol is A at the end, what should come after A? If A comes after A,

If multiple A's are there, is there any problem? No. If it is there at the end, it is A. It ends with A. Now, if I have A here. If there is one more A after this. As many A's as you want. There is no problem. Because, A should be there at the end. So, its logic is simple.

Q1 is self-duplicated on A. That's it. On B. If I am on B. Now, look. Q1. If I am on B, let's suppose. I put it in the trash state. I put it in the trash state. So, what is the problem? What can be the problem? If I have a string like this. A. B. A. A, B, A. So we discussed that A, B, A should be accepted because A, B, A ends with A. But, if I run this here starting with Q0, I ran A from Q0 to Q1, B comes to Q1, B comes to Q0, Q2. Now, where will the next A go? Obviously, the trap state always goes to itself.

So, you will get stuck on Q0 and Q2. You will get stuck on Q2 because you made a trap. So, there is no need to make a trap here because A, B, A type strings should also be accepted. So, what should you do to accept this? To accept this, you don't have to make a trap state.

Summary

**Introduction**

This lecture from Geeksmashers focuses on designing a Deterministic Finite Automaton (DFA) that accepts all strings starting with 'A' and ending with 'B'. The speaker emphasizes that DFA design is primarily logical rather than theoretical, making it an important topic for academic and competitive exams.

**Problem Statement**

The language consists of all strings that:

* Must start with the character 'A'
* Must end with the character 'B'
* The minimum valid string is "AB"

**Design Approach**

The speaker outlines a methodical approach to DFA design:

1. First identify the minimum valid string ("AB")
2. Create initial states to handle the first 'A' (Q0→Q1) and final 'B' (Q1→Q2)
3. Make Q2 a final state
4. Design appropriate transitions for all possible inputs at each state
5. Handle edge cases like multiple consecutive 'A's

**Common Design Mistakes**

The lecturer explains several potential errors in the design process:

* Trapping on receiving 'B' at the start state (Q0)
* Trapping on receiving another 'A' after the first 'A' (at Q1)
  + This would incorrectly reject strings like "AAB"
* Creating a path back to the start state after receiving 'B'
  + This would incorrectly handle strings like "ABA"

**Educational Importance**

The speaker highlights that DFA design is:

* A crucial topic in Theory of Computation (TOC)
* Frequently tested in university and competitive exams
* Best learned through practicing multiple examples
* Primarily about developing logical thinking skills rather than memorizing theory

Notes

Transcript

Hello friends, welcome to Geeksmashers. Next example is Constructor DFA which accepts a language of all strings which must start with A and end with B. That means all the strings in the language should start with A and end with B. If you want to design DFA in this way, then you have to follow the same method that I have already discussed.

And why is this important? Because in all the competitive exams, if we talk about your college or university level exams, DFA design is one of the most important topic of TOC. And the more examples you do, the more benefits you will get. Otherwise, you will find logic somewhere. Because this is a logical question. There is no theoretical concept behind this. You have to apply logic.

And logically, you can solve this easily. So this is very important for your academic and competitive exams. So again, the first method is that you have to write the language. Now, which languages will come? What are the strings that should be starting with A and ending with B? Minimum to minimum string will be A and B. Because it can't be less than A and B. If you write, let's suppose, epsilon. Then neither A nor B comes in epsilon. If you write it alone, then it is partially true.

It starts with A and ends with A and B. But we have a question. It starts with A and ends with B. B alone can't come. So, what will be the minimum to minimum? A and B. So, you can make as many questions as you want. Like A, B, A, B, A, B, A, B, A, B, A, B.

You can make as many questions as you want. Apart from this, let's suppose if I write A multiple times.

Getting a shipping quote just got so much easier. Your competitive shipping rate, lightning fast. If you apply single B, you will reach Q2. And what do you make of Q2? You make it a final state. So look, A and B are set. A and B. You have to make single A. You don't have to make a loop. Because if you make a loop, it means multiple times. But first of all, you have to set the minimum string course.

So which is the minimum? A and B. Now what is the simplest step? Fill in the blanks. Fill in the blanks means, Q0 went to A. Where will it go to B? Where will Q1 go? And where will Q2 go to A and B? You just have to fill in the blanks. Now look at Q0 here.

A should come in the starting. But if B comes in the starting, then what should happen?

It will automatically go to A and B. What does that mean? It can't go anywhere else. You have given it a punishment. You can't go anywhere from here. Because it will automatically go to A and B. As soon as B comes. Let's say as soon as B comes. As soon as B comes, you have tapped it.

Now it can't leave from there. It can't go anywhere. Because it will automatically go to A and B. So you have filled this Q note. You have filled it on A and B. Now let's talk about Q1. Q1 will go to B. But it can't go to A. You can either tap it on A.

If you trap it on A, can there be a problem? Yes, there can be a problem. How? Let's suppose if you trap it on A and the string is AAB, can there be a problem? Yes, there can be a problem. Can AAB come? Yes. Can AABB come? Yes, there can be a multiple.

So, starting with A, ending with B, fine. But, A, A. See, how do you run it? First, A came here. Second, A came, you reached the trap. B, B. So, where did you get stuck? You got stuck in the trap state. So, this didn't get accepted. But, this should get accepted.

So, your DFA is not the right design. So if you don't put it in the trap state, then what will you do? What are the other options? What are the other options you have? You can send it back. If you send it back, can there be a problem? If you send it back, obviously you don't have to make a loop to send it back.

What do you do directly? Make a loop right here. Simple. If you make a loop right here, So let's suppose if I talk, A, A, A, B. If you make it this way, Let's suppose A, A, A, B. So if you send it back, If you haven't made a loop, and if you send it back, what will happen? A, A, A, A, A, and B. This will be accepted. But let's say you put A four times. If you put A four times, then A, A, A, and A. And next comes B. As soon as the next B comes, obviously, this won't be accepted.

How? It's simple, you have to run. A single comes, next A comes, you go here. Third A comes, you go here. Fourth A comes, you go here. Now next comes B. If B comes, you will go to the trap string. Obviously, you will go to the trap string. But if you look at this string carefully,

Starting with A, yes. Ending with B, yes. So, this should be accepted. But this is not accepted. Why? Because you sent it back. You sent A back. So, instead of sending it back, what do you do? Make it a loop. Because we don't have any problem with A, as we will get B in the end.

Now, look. A, A, A, A, B. So, this string number gets accepted. So, you just have to apply this logic. So, your DFA will be designed easily. Last, you have Q2 left. Now, look at Q2. Q1 will be final on A and B. Q2 is for B. It is a simple logic for B.

And what did you do? You put it in the trap state. Now what will happen? A, B, A, B. So this is not accepted. But A, B, A should also be accepted. Because the beginning is A and the end is B. So that means you can't put it in the trap. What is the second option?

You send it back to Q0. If you send it back to Q0, then what will happen? A, B, A, B. This is also not accepted. A, B, A should also be accepted. But A, B, A and B. This is not accepted. So that means you don't have to apply this logic either.

Summary

**Main Problem**

* Construct a Deterministic Finite Automaton (DFA) that accepts all strings either not starting with 'A' or not ending with 'B'
* This involves using complementation and set operations to solve the problem

**Solution Approach**

* The language can be expressed as: "not starting with A OR not ending with B"
* Using De Morgan's laws, this equals the complement of: "starting with A AND ending with B"
* Steps:
  1. First create a DFA for "strings starting with A and ending with B"
  2. Then complement this DFA by making all final states non-final and all non-final states final

**Implementation Details**

* Original DFA for "starting with A and ending with B" had:
  + Initial state Q0 (non-final)
  + State Q1 reached after reading 'A' (non-final)
  + State Q2 reached after reading 'B' from Q1 (final state)
* After complementation:
  + Q0 becomes final
  + Q1 becomes final
  + Q2 becomes non-final

**Examples and Verification**

* The resulting DFA correctly accepts:
  + Empty string (epsilon) - accepted because it neither starts with A nor ends with B
  + 'A' - accepted because it doesn't end with B
  + 'B' - accepted because it doesn't start with A
  + 'BA' - accepted because it doesn't start with A
* The DFA correctly rejects:
  + 'AB' - rejected because it both starts with A and ends with B

Notes

Transcript

Hello friends, Welcome to Great Smashes Next example is Construct a DFA which accepts a language of all strings not starting with A or not ending with B In the last video, we discussed about containing A or at least A starting with A, ending with A, starting with A and ending with B Questions like these we discussed But in this video, we will discuss about not doing it Not starting with A, not ending with B First of all, if we write its language Then which form will we get in the language?

Not starting with A I am not ending with B. So if I talk about lambda or if I talk about epsilon, I mean, none. So see, lambda or null will also be accepted in this. Because neither does it start with A nor does it end with B. Apart from this, if we talk about, let's suppose, only A or only B.

If I talk about B, A. I mean, you have multiple strings being made here. But let me tell you that whenever...

Not starting with A Let's suppose it is A statement Not ending with B Let's suppose I consider this whole statement to be B So this is my B Not starting with A means A, not ending with B means B And what else is there in between? Let's calculate the other one

That's it. So here, if I take its complement, then whose complement will it be? A complement intersection B complement. What does A complement mean? A means not starting with A. So what will its complement be? Starting with the A. What will its complement be?

Starting with the A. What is the meaning of Union and Intersection? A What is the meaning of B? Not ending with B What is the meaning of B? Not ending with B What is the meaning of B? Starting with A and ending with B We discussed this in the last video How do we design BFA?

Which is accepting all the strings Starting with A and ending with B We discussed this in the last video I am making it here Why do I have a starting state? We went to Q1 on A and we went to Q2 on B. And what was Q2? It was my final state. If you haven't checked this video, then do check it out. The link is given in the description box. Starting with A and ending with B, how will we design it? I am using it directly here.

A is being looped here. B is being looped here. And it is coming back for A. And what did we do for B here? We put it in the trap. Starting with A and ending with B So this is our starting with A

A and ending with B. But I don't want to make it starting with A and ending with B. I want to make it not starting with A and not ending with B. So what we have to do here is take its complement. Because if you take its complement, then this will come.

And if you take its complement, then this will come. So obviously these two are vice versa. If you want to take this as a compliment, then how do we take DFA as a compliment? Taking DFA as a compliment is a very simple method. The final states, all the final states, make them non-final. And all the non-final states, make them final states. So see here, what is non-final?

Q0. So make Q0 final. Q1 is non-final here, so make it final. Q2 was the final state here, so I make it non-final. So this is my... Not starting with A and not ending with B's DFA. So, if you get Apsilon alone, will it be accepted? Yes, because this is my starting state. So, if you don't give A or B in the starting state, what is Epsilon B doing? It is accepting. Will A be accepted alone?

Yes, A will be accepted alone. Why is it happening alone? Not starting with A and not ending with B. So, this is starting with A. But there is an OR in between. Not ending with B. Is this true? Yes, this is true. Not ending with B. So, see, it is simple for you here.

True or false? The answer will be obviously true. I mean, you have to check it in a simple way. If you take A alone, then not starting with A. This is false, but it's true. Not ending with B, so obviously it's not ending with B. If I take B alone, this will also be accepted. It should be. Yes, obviously it should be. Why? Not starting with A. See, B is not starting with A. And if there's more in the middle, then our next part of checking, if it's true in one part, then this string will be accepted.

Then if I talk about A, B. Not starting with A and not ending with B. So if you take A, B, A, B. A, B won't be accepted here. Why? Because it starts with A and ends with B. So A, B won't be accepted here. So my DFA is working properly. Apart from this, you can take any other. If you take B, A, then this is also accepted.

Not starting with A and not ending with B. So yes, all the strings will be accepted. Except...

Summary

**Problem Statement**

* Design a DFA that accepts all strings over {0,1} where:
  + 2nd symbol must be 0
  + 4th symbol must be 1
  + Minimum string length is 4

**Key Principles**

* Minimum states formula: n+1 (where n is minimum string length)
* With trap state: n+2 states
* For this problem: minimum 5 states, 6 with trap state
* Every state needs transitions for all possible inputs (0,1)

**Design Process**

* Start with state Q0 (initial state)
* Q0 accepts both 0 and 1, transitions to Q1
* Q1 must see 0 to proceed to Q2
* If Q1 sees 1, transition to trap state Q5
* Q2 accepts both 0 and 1, transitions to Q3
* Q3 must see 1 to proceed to Q4 (final state)
* If Q3 sees 0, transition to trap state
* Q4 (final state) loops on both 0 and 1
* Q5 (trap state) loops on both 0 and 1

**Implementation Notes**

* After meeting the required conditions (2nd=0, 4th=1), the DFA remains in the accepting state
* Trap states are used when input violates the required pattern
* Complete DFA needs transitions for all input symbols from every state

Notes

Transcript

Hello friends, welcome to Geek Special. Next example on DFA designing is Design a DFA which accepts all strings over 0 and 1 Its 2nd symbol must be 0 and 4th symbol must be 1 Means we have to design a DFA which accepts all those strings In which the 2nd symbol should be 0 and 4th symbol should be 1 So whenever we design a DFA, first of all we have to find the minimum string

So, the first position is fixed, but the letter can be 0. The second position is fixed, but the letter can be 0. The third position is fixed, but the letter can be 0 or 1. The fourth position is fixed, but the letter can be 1. So, let's start. First of all, let me tell you a point.

Whenever there are questions like this, in which you will be given a length, or whatever DFA you are designing, this string should be set, like 001, 100, 0001. Questions like this come. So, the minimum length is 4. So, in DFA, what will be the minimum states?

n plus 1. Keep this point in your mind. I am talking about minimum state, not maximum. Whenever you design a DF, the minimum state will be n plus 1. And what is n here? n is the length of the minimum string. Like in this question, we discussed that the length of the minimum string is 4.

So, how many states will be less? 4 plus 1, that is 5. These are the minimum states. So, let's start with Q0. That is our starting state. On Q0, we have... We have an option on Q dot. Either 0 or 1 can come. So I define 0 or 1 here. You don't have to make a loop here. If you make a loop, then you are inviting multiple 0s or multiple 1s. But you don't have to make a loop because we have a fix. Only one symbol will come.

Either 0 or 1 can come. So here we have said that if 0 comes on Q dot, then I will go to transition Q1. If 1 comes on Q dot, then also I will go to transition Q1. Next we have Q1. My second symbol is fixed. That is 0. As soon as 0 comes, you will get up.

You will reach Q2. Now on Q2, you have the third position again. We have fixed it. But the letter can be 0 or 1. So where will you reach? You will reach Q3 state. And if you apply 1 on Q3, you will reach Q4 state. That is your final state. So see, if we talk about any letter here, you have a choice of 0 or 1. We have fixed the second 0.

You can get 0 or 1 on the third position. We have fixed the fourth. That is 1. So this is how you have to design DFA. So see, we had to find the minimum states. 5. 1, 2, 3, 4, 5. But is this DFA complete? No. When will DFA be complete? When the input symbol is complete in each state. For every input symbol, you have to make a transition. So we made a transition for 0 and 1 on Q1. But we made a transition for 1 on Q1. If I make a loop for 1 here, then you can get 0, 1, 1, 1, 1, 0, 0, any string of this type.

The reason is that multiple times 1 can come if you make a loop. So we don't have to make a loop. What we have to do is fix it. To fix it, you have to use the staff state. If the second symbol is 1, So you can trap it. If the second symbol is 1, then you can trap it. Because the second symbol should be 0. If it is 0, then you will reach Q2. But if it is 1, then let's suppose you will reach Q5. And Q5 is your kind of trap state. Then, look at Q2.

Now for 0 and 1, Q1 is fine. Q2 is final for 0 and 1. That means you made transition for 0 and 1. Q3, you made it for 1. Q4 is fine. But where is it going for 0? For 0, obviously, if you make a loop, the same problem will arise. If you make a loop for 0, then multiply time 0 can come in this position.

So what you have to do for one time, or as soon as your 0 comes, put it in the trap state. Because we don't want your 0 to come. We want 1 to come, then it's fine. If 0 comes, then put it in the trap state. So Q5 is your trap state. So Q3 will be fine on 1 and 0 as well.

Next, let's come to Q4. Now where will Q4 go for 0 and 1? Q4 is a simple logic. Make a loop for 0 and 1. What is the reason for that? Second symbol is 0. Fourth symbol is 1. And what comes after that? After that, something happens. I mean, after the fourth symbol, something happens.

0 comes, 1 comes, 0, 0 comes, 1, 1, 1 comes. We have fulfilled our condition of 2nd symbol and 4th value. Next time we don't have any problem. So 0 and 1 are fixed. There is no problem for wrap state. Wrap state is always for 0 and 1. That's why it is called wrap state. So you have to design DFA in this way.

What are the number of states? 1, 2, 3, 4, 5, 6. So here we have done formula analysis. Minimum number of states are 7 and plus 1. In DFA, we generally add one more one and that one we add for trap state. So generally we don't need trap state in every state or in every condition. But generally we have trap state in DFA. If you are including trap state, then minimum number of states you will get is n plus 2.

If you don't want to include trap state, then minimum number of states you will get is n plus 1. So trap state can be multiplied. एक टैप स्टेट से ही आपका काम चलेगा.

Summary

**Problem Overview**

The lecture explains how to construct a Deterministic Finite Automaton (DFA) that accepts binary strings divisible by 3. This is part of a general approach for creating DFAs that recognize numbers divisible by any integer.

**DFA Construction Approach**

* When dividing by 3, there are three possible remainders: 0, 1, and 2
* These remainders determine the number of states in the DFA
* Three states are needed: Q0, Q1, and Q2
  + Q0: represents numbers with remainder 0 (divisible by 3)
  + Q1: represents numbers with remainder 1
  + Q2: represents numbers with remainder 2

**DFA Structure**

* Q0 is both the starting state and final/accepting state
* Transitions between states are based on reading binary digits (0 or 1)
* For each state, transitions are determined by calculating what remainder would result after appending a 0 or 1

**State Transitions**

* From Q0 (remainder 0):
  + Reading 0 keeps it in Q0
  + Reading 1 moves to Q1
* From Q1 (remainder 1):
  + Reading 0 moves to Q2
  + Reading 1 has transitions explained but not clearly specified
* From Q2 (remainder 2):
  + Transitions were being explained but not fully specified in the transcript

**Examples Discussed**

* Binary string "001" (decimal 1) should end in state Q1
* Binary string "101" (decimal 5) should end in state Q2
* Binary string "100" (decimal 4) should end in state Q1
* Multiple other examples were partially discussed to demonstrate the DFA's operation

Notes

Transcript

Next example is Constructor DFA which accepts the language of all binary strings divisible by 3 over the alphabets 0 and 1. So whenever you get a question like divisible by 3, divisible by 2, divisible by 5 or whatever, there is a simple method to answer these types of questions.

First, you check that if you divide any number by 3, then what will be the remainder? So if you divide any number, obviously one will be the remainder. One will be the remainder and the remainder can be 2. For example, if I divide 3 by 3, the remainder will be 0.

If I divide 4 by 3, it will remain 1. If I divide 5 by 3, it will remain 2. And if I divide 6 by 6, it will remain 0. This is how our loop works. So I have 0, 1, 2. What is the significance of making this? What is the use of this? You will get to know from this.

You will get to know how many states I will have here. Not exactly states, but at least you will get to know that we will have so many states. So one of your state will be Q0.

If you divide that number by 3, and if it is exactly divisible, i.e. the remainder is 0, then you want it to end at Q0. If you want to divide any number and the remainder is 1, then you want the string to end at Q1. And if you divide any number by 3 and the remainder is 2, i.e.

if you divide 5 by 3 and the remainder is 2, then you want the string to end at Q2. You can start this with a simple example. First of all, let's suppose that I have Qnode is my starting state. Qnode represents my remaining state. If you make a binary state, 0, 0, 0, 1, 1, 0, 1, 1, you can make it multiple times. 0 is divisible by any number. Qnode is my final state.

If your 1 comes here, then what do you want as soon as 1 comes? As soon as 1 comes, you want to take it to Q1 state. Means, let's suppose we have 0, 0, 0, then obviously it is set. 0, 0, 1, means the next thing that comes, if let's suppose only 1 comes or 0, 1 comes.

What is 0, 1 in decimal? It is 1 only. So if you divide 1 by 3, then what will be your answer? It will be 1 only. So see, 0, 1. As soon as 1 comes, you take it to Q1 state. Why? So that whatever 1 comes, in whichever remainder 1 comes, it should end up at Q1.

What is the remainder we have next? Next remainder will be 2, because one remainder is 0, one is 1 and one is 2. And how do we represent 2 in binding? We represent 2 with 1 and 0. So 1 is already there, you can make 0 more. So this is your Q2 state. Whenever your remainder is 1 and 0, means whenever your remainder is 2, it will always end at Q2.

For example, let's suppose you take 5. What is 5? 1, 0, 1. Now see, 1, 0. You want to end at 2. To end at Q2, obviously you have moved Q0 from 1 to 0.

So how will you be able to do Q1? First you will go to Q1. Now if you make a cell phone here or take it here, then obviously it will not terminate, it will not be accepted. If you bring this 1 here, then it will be accepted. Why? 1, 1. See, it is accepted.

Similarly, let's suppose 1, 1, 1, 1. What happens in 1, 1, 1, 1?

Yes, it is also readable. So, 1, 1, 1, 1. So, look, this is also set. So, your Q0 has gone to 0. It has gone to 1, fine. Q1 has gone to 0. It has gone to 1. This is also fine. Q2. Now, you have to complete Q2. So, to complete Q2, here, if I write let's suppose 1 first.

1, which I took as an example earlier. 1, 0, 1. What is 1, 0, 1? 5. Now, if you divide 5 by 3, what will be the remainder? 2. That is, you want that 1, 0, 1 should end at Q2. So, how will you do it? Assume that it ends here.

Let's suppose you take string 1 0 0. What is 1 0 0? It is 4. Now divide 1 0 0 by 3. Let's suppose 1 0 0. That is 4. Divide by 3. Then you will get 1. So you want to end it at Q1. So how will you do it? 1 0 0. So for 0, what will you do? Take it to Q1.

So that 1 0 0 comes. That is 4 divided by 3. So it should end here. Now you can make any string.

Summary

**Definition and Components**

* NDFA (Non-deterministic Finite Automata) is defined by five components:
  + Q: Set of finite states (e.g., Q0, Q1, Q2)
  + Sigma (Σ): Input symbols/alphabet (e.g., 0, 1 or a, b, c)
  + Q0: Starting state
  + F: Set of final states (subset of Q)
  + Delta (δ): Transition function

**Key Difference from DFA**

* In Deterministic Finite Automata (DFA): For any state and input, there is exactly one next state
* In Non-deterministic Finite Automata (NDFA): For any state and input, there can be:
  + Multiple possible next states
  + No next state (null state)
  + Example: From state Q0 with input 1, you might reach Q1 or stay at Q0

**Transition Function**

* DFA: δ: Q × Σ → Q (maps to a single state)
* NDFA: δ: Q × Σ → 2^Q (maps to power set of Q)
* With |Q| states, NDFA has 2^|Q| possible transitions
* Example: With 3 states, there are 2^3 = 8 possible transition outcomes

**Educational Context**

* Important concept for competitive exams and university-level computer science courses
* Foundation for understanding automata theory

Notes

Transcript

That is non-deterministic finite automata. This is also called NFA. This is very important for your competitive exams, even for your college or university level exams. So guys, quickly like the video and subscribe to the channel if you haven't already.

And please press the bell button so that you get all the latest notifications. So let's start. NDFA, that is non-deterministic finite automata. When we define NDFA in this, the definition is almost the same as the one we designed in DFA. That is deterministic finite automata.

So we define DFA as well. In these five doubles, we have made the same doubles. First of all, we have Q, that is set of finite states. Set of finite states. That's why the name is non-deterministic finite automata. The number of states in the machine is finite. So this is Q, that is set of finite states.

For example, Q0, Q1, Q2, we have three states. So the number of states is three. So you can say their names. Because the whole set is Q. Then we have Sigma. What is the meaning of Input Symbol? Sometimes we take it in the form of 0,1 or ABC. That is your alphabet also. We take the alphabet here. That is your Sigma. Then Q0. Q0 is the starting state. Which is the starting state of your machine? That is known as Q0.

Then F. F is the set of final states. What is the meaning of final states? Generally we have only one final state. But there can be multiple final states. But all the final states will always be a subset of Q. Because Q is the set of all final states.

Let's say I applied 0. Now after applying 0, where will you reach? Let's suppose I am reaching Q1. I have fixed that if I apply 0 from Q0, I will reach Q1. If I am only reaching Q1, there is only one choice. There is no other choice. There is only one option that you will reach Q1.

That is called deterministic. This is deterministic behavior. But in the case of NDFA, we have non-deterministic behavior. What does non-deterministic mean? It can also happen that you reach Q1. Q2 Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15 Q16 Q17 Q18 Q19 Q20 Q21 Q22 Q23 Q24 Q25 Q26

For that, I can use both DFA and NFA. It is easy to design DFA. But we will discuss the difference and similarity in the next video. Here, we are discussing the basic introduction of SIRF and SIRF-X. So, you will be clear from this example. Q0. You applied one signal on Q0.

You gave one input. As soon as you give one input, you will reach a state of Q1. And you will have a choice to stay on Q0. You will have multiple choices here. Q2 is Q0. Where is it going for 0? We did not define it for 0. You can say that it is going to a null state for 0.

Q1 is going to 0 and Q2 as well. So we have multiple choices. That is why it is known as non-deterministic finite automata. So how do we define delta in DFA? We define delta as Q into sigma. That is Q. Q into sigma always denotes Q. Q is the transition function that we know.

It is one of these. For example, if we take an example of DFA or NFA, let's suppose it has three states. Q0, Q1 and Q2. Q1, Q2. And here we have sigma. What do we have in sigma? 0 and 1. Input symbol is 0 and 1. Now here in the case of DFA, what do we have? Q. If you multiply Q0 by 0, then Q0 is 0. Q0 is 1. Q1 is 0. Q1 is 1. Q2 is 0. Q2 is 1. I have multiple choices here.

Now see, what do I have here? Q. In the case of DFA, what do we have? Q. Q means either Q0 or Q1 or Q2. I mean, in one of these, your transition must have happened. There is only one move. But here, in the case of NDFA, how many moves do I have? 2 raised to power q. That is power set of q. Now, how much is q in this question? Let's suppose we have a number of states of 3. That means you have 2 raised to power 3 choices. 2 raised to power 3 transitions.

That is 8. And which are these 8 transitions? It is simple. 8 transitions will be made. The first transition is 5. Meaning, null state. We are not going to any state. The second one can be q1, q2, q3. Sorry, we have taken q0 here. So take q0. Simple.

If you apply 0 on Q1, you will either end up on Q1 or you will end up on Q2. Similarly, if you apply 0 on Q2, you will end up on Q2. You have multiple choices here. Let's say 0. In the case of 0, it is not going anywhere. In the case of 0, it is in the null state.

The important point we have in the case of NDFA is that how many transitions do we have? Q into sigma is not equal to Q. It is equal to Q in the case of DFA. It will come in the case of NDFA. That's why there are multiple moves, there are multiple choices. Thank you.

Summary

**Introduction**

This video from Geeksmashers explains the key differences between Deterministic Finite Automata (DFA) and Non-deterministic Finite Automata (NFA), concepts important for computer science exams. The presenter mentions having previously created videos on how to design these automata, with links available in the description.

**Key Differences**

* **Transition Function (Delta)**: The primary difference lies in how transitions work
  + DFA: Each state-input pair leads to exactly one state
  + NFA: Each state-input pair can lead to multiple states (multiple choices)
* **Dead Configuration**:
  + DFA: Not allowed - transitions must be defined for all input symbols from every state
  + NFA: Allowed - transitions can be left undefined for certain inputs
* **Epsilon/Null Moves**:
  + DFA: Not possible
  + NFA: Allowed (transitions without consuming input)
* **Real-world Applications**:
  + Digital computers are deterministic, resembling DFA
  + Both DFA and NFA are used to check if languages are regular

**Complexity Considerations**

* **Design and Understanding**:
  + DFA: Generally more difficult to design and understand, especially for complex cases
  + NFA: Easier to design and understand due to fewer states and more flexible rules
* **State Conversion**:
  + Converting NFA to DFA can result in up to 2^n states (where n is the number of NFA states)
  + This exponential growth makes DFA potentially more complex

**Purpose**

Both automata types are used to determine whether languages are regular, though they differ in practical implementation and complexity.

Notes

Transcript

Hello friends, welcome to Geeksmashers. In this video, I am going to explain the difference between DFA and NFA. Which is very important for your cognitive exams and even for your college and university level exams. So guys, quickly like the video and subscribe to the channel if you haven't already.

And please press the bell button so that you get all the latest notifications. So let's start with the difference between DFA, that is deterministic finite automata, and NDFA or NFA, that is non-deterministic finite automata. First of all, let me tell you, what is introduction to DFA?

How is it designed? And similarly, what is introduction to NFA? How is it designed? I have already made a video on this topic and its link is given in the description box below. So do check those videos as well so that you get to know how we design, what happens and how we define it. Then you will be able to properly clear the difference between the definitions.

So let's start. First of all, dead configuration is not allowed. First of all, let me tell you that the major difference between DFA and NFA is that of delta. What is delta? Transition function. Transition. Transition means that if you are in a state, let's say Q0, and if you apply any input symbol like 0, 1, A, B, any alphabet, then at what state will you reach? Whether you reach Q1, Q2, Qn, or yourself, wherever you reach, that is called a transition.

So the major difference between the two is the difference of transition. So the first difference is that dead configuration is not allowed in DFA. What does dead configuration mean? Let's suppose you made DFA. Q0, you applied 0, you applied 0 input, as soon as you applied, you reached Q1. But where are you going for 1? I mean, where is Q0 going for 1? Neither did you make a self-group for 1, nor did you define 1 in any other state. That is called a dead. I mean, you did not define 1 that where is this state going for 1?

I mean, where is any state going for 1 or 0? This is called a dead configuration. For 0, 1 or any other input symbol, a transition has been made. So this type of option is not allowed in DFA. If you have defined for 0, then you have to define for 1 as well. Whether you make a cell clue for 1, or for 1 here, or anywhere. But you have to define for all the input symbols.

You have to define for all the input symbols, 0, 1, etc. You have to define for any state. Similarly, you have to define for Q1 as well. If you are going from Q0 to Q1, then you are going from Q0 to Q1. Now, you can't define DFA in this way that you are going from 0 to Q1, Q0 and Q1. If you apply one state to one state, then you should have only one state. It can be any state, but there is a single option.

But in case of NFA, you have multiple options, multiple choices that you can go to multiple states. So, this is the major difference between DFA and NFA. Thirdly, that is absilem. Meaning, null move is allowed in case of NFA. Possible in NFA, not possible in BFA. Then, digital computers. If someone asks you, what is their real life application? So, all the digital computers are basically deterministic. And let me tell you why we use both of them equally. To check whether the language is regular or not.

I mean, we use DFA and NFA for regular languages. But, if we talk about the real life example, digital computers, they are all deterministic. I mean, all the real machines, real world machines, they are deterministic. They will tell you that if you give one input, then you will reach the corresponding state of that input.

It can be any state, but it will be a single state. You do not have the option of multiple states or multiple choices. Which is there in case of NFA. So, no deterministic feature is not associated with NFA. They are actually deterministic, they are not non-deterministic. Lastly, designing and understanding is difficult in case of DFA. Not all the cases, but some cases are like this. You can say that the strings are lengthy. You can say that designing is difficult.

What does designing mean? You have to define for every state and symbol. So, number of states are more jungle. In case of DFA, number of states are more. Obviously, it will be difficult to design and understand. As compared to the NFA, because it also has a null mode. You are also allowing dead configuration in this. So, if you are allowing dead configuration, it may be that you are not defining anything for symbols, that from where you are going. So, obviously, the number of states will be less in case of NFA.

So, you can say that designing is a bit easier as compared to the DFA. And understanding is also easier as compared to the DFA. And in the last point, I will tell you that if you have an NFA, and if you want to convert NFA to DFA, if you want to convert any NFA to DFA, then the maximum possible states can be 2 raised to the power n.

If you have made an NFA, and the number of states in NFA is n, then if you are making a DFA, then the maximum possible value can be less than that, but the maximum is 2 raised to the power n states possible, which is our exponential number. So, obviously, you can say that if the number of states is so high, then it becomes a little complex to design and understand it as compared to the NFA.

So, these are all the major differences between DFA and NDF. Thank you.

Summary

**Problem Statement**

* Design a Non-deterministic Finite Automata (NFA) for all binary strings where the second-last symbol is 1
* Using binary alphabet (0 and 1)
* Minimum string length is 2 (examples: 10, 11)

**NFA Design Process**

* NFA requires at least 3 states (n+1 = 2+1 = 3)
* Initial state has a self-loop for both 0 and 1 inputs (representing any number of initial symbols)
* From initial state, a transition on input 1 leads to second state
* From second state, transitions for both 0 and 1 lead to final state
* Dead configurations are allowed in NFA (no need to define transitions for every input at every state)

**NFA vs DFA Comparison**

* NFA design is simpler for this problem
* DFA would require more states and higher complexity
* Converting this NFA to DFA would require using a conversion algorithm
* DFA doesn't allow dead configurations

**Reference**

* [ ] Check previous video explaining differences between NFA and DFA

Notes

Transcript

NFA of all binary strings In this video, we are going to design an NFA that is non-deterministic finite automata. Let's discuss an example. NFA of all binary strings in which second last symbol is 1. I will tell you that our alphabets

It doesn't matter if you are given 0, a, b, or alphabet, the concept is the same. You just have to check in which form the question is given. So in this question, we are given a binary string in which the second to last bit should be 1. So if we want to design this in NFA, then first of all...

In the last bit, I have an option that anything can come out of 0 and 1. If we design its language, how will you design the language? In the language, we should have the second last bit 1. The second last bit means...

Let's suppose this is my second last place. There should be 1 in the second last place. And in the last place, anything can come out of 0 or 1. So you can write this as 0 plus 1. What does plus mean? It means either 0 or 1. But any one of the two symbols will come out of 0 or 1.

And what should come before that? It should come before that. 1. And what should come before 1? Anything can come before 1. What does anything mean? As many 0s as you want, as many 1s as you want. Any combination of 0 and 1 can come. So how do we denote any combination of 0 and 1?

0 plus 1. Why not? The second last symbol should be 1. Obviously, I can add minimum to minimum string. Minimum to minimum string will be 1, 0 or 1, 1. What will be the minimum length? Minimum length is either 1, 0 or 1, 1. Because the second last is 1. But anyone can come from 0 or 1 at the end.

So, what is the length of the minimum string? 2. If the length of the minimum string is 2, then obviously the number of states will be n plus 1. That is, how many bits are there? 2. What is the length? 2. So, 2 plus 1 will be your minimum 3 states. So, let's make minimum 3 states.

So, I have closed the second last symbol. It has been edited to include proper punctuation.

After this, what should I have? For 0 and 1, we design this only for DFA. Because in DFA, I have to design for every state, for every input. Where is it going for 1, where is it going for 0, where is it going for 0, where is it going for 1, where is it going for 0, where is it going for 1.

In the case of NFA, there is no problem. Dead proliferation is allowed. Dead proliferation means that you don't have to make 0 or 1 here. Nor do you have to make any trap state. You can make it simple like this. But what we have is that anyone can come in 0 and 1 in the starting.

So if anyone can come in 0 and 1, what do I make of 0 and 1?

I'll tell you what will happen. This is your expression. Anyone can come from 0 or 1 in starting. So I made a loop in 0 or 1. What is this loop making? Anything can come from 0 or 1. It can come at any time. But after that, you should have 1. And after 1, any single symbol should come from 0 or 1.

So this is how we design MFA.

So there are multiple choices in this. For 1, it is in the loop and for 1, it is going to Q1. Here, if you talk about Q2, where is Q2 going to 0 and 1? It is not going anywhere. So this is called NFA. Anyway, the difference between NFA and DFA, I have explained that in the last video.

If you have not checked that video, then please do check it. Because all the videos I have brought from TOC, I have brought from the initial state. So all these videos are connected to each other. So do check that video once so that the difference between these two is clear.

So this is how we design NFA. But if you want to design DFA corresponding to this, So to design DFA, you will have to use a conversion algorithm and convert it from LFA to DFA. And if you convert and make DFA, then you will know why it is easy to design LFA. Because in DFA design, you will have a little complexity. The number of states will be more and the complexity will be more because there is no dead configuration allowed.

Now multiple choice is allowed. Thank you.

Summary

**Overview**

This lecture explains the conversion process from Non-deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA) using an example of binary strings where the second-to-last bit is 1. The lecturer builds upon a previous explanation of the NFA design for this language.

**Conversion Process**

* Start with the NFA transition table containing states Q0, Q1, Q2 (with Q2 as the final state)
* Copy the first row of the NFA table (start state) as is
* For each new state combination that appears:
  + Determine transitions for input symbols 0 and 1
  + Create new state combinations as needed
  + Ignore "dead" configurations
* The start state in DFA is the same as in NFA
* Final states in DFA are any state combinations that contain a final state from the NFA (Q2 in this example)

**Key Insights**

* DFA design is more complex compared to NFA design
* DFA diagrams tend to be more complicated than NFA diagrams
* DFA will have more states than the corresponding NFA
* If an NFA has n states, the corresponding DFA can have up to 2^n states
* Unlike NFAs, DFAs cannot have choices or dead configurations
* For every input symbol, DFAs must specify exactly where to go next

Notes

Transcript

I am going to explain how to convert NFA to DFA, i.e. how to convert non-deterministic finite automata to deterministic finite automata. To explain this, I am going to take an example. The example is NFA of all binary strings in which the second to last bit is 1.

I have explained this example in the last video, how we design NFA which accepts all the strings and the second to last bit should be 1. So, this becomes NFA and you have to make the corresponding transition table. The transition table is a very simple way.

To make the transition table, you have to Q0, Q1, Q2

Transition Table Easy to convert from NFA to DFA

To convert in DFA, we will make a transition table of DFA. To make a transition table of DFA, what you have to do is simple. Q0, the first line denotes the start state of NFA, Q0. And this round denotes the final state of NFA, that is Q2. This is important because when we convert this in DFA, this point will help you there.

So copy the first row as it is. The algorithm is very simple. Copy the first row as it is. So if I copy the first row, So for 0 and 1, I have Q0 here. Q0 Q1. Copy the first row as it is. Now you have to extend this row further. Look, for Q0, you know that Q0 and Q1 are making a transition here. So Q0 is already completed. What state do I have next? Q0 Q1. Q0 Q1 is your next state.

Look at this state. Q0 Q1 is your next state. But Q0 Q1 is going anywhere. I mean, where is its transition in the case of 0 and 1? You have to check that from here. Copy the first row as it is. We made a transition for Q0. Look here, for 0 and 1, Q0 is here.

If it was Q1, then I would have expanded Q1. But here it was just Q0, so we have already expanded Q0. Next one is Q0 Q1. Now we are expanding Q0 Q1. That is, where is Q0 Q1 going for 0 and 1? You have to check that on the transition table. Look, Q0 Q1.

In this way, you keep your finger on both Q0 Q1. Where is it going to 0? It is going to Q0 and Q2. So you write both. It is going to Q0 and Q2.

Q0 and Q1 Keep your fingers like this Where is it going on 0? On Q0 and Q2 So write both states You write Q2 first, write Q0 It doesn't matter here The order doesn't matter You just have to write both states Where is it going on 1? It is going on Q0, Q1 and Q2 Means it is going on all three So you write all three Q0, Q1, Q2 You have to write like this On 1, these three states will be covered On 0 and 1 I wrote these two states on 0 On 1, these three states will be covered Q0 and Q1 On 1, Q0, Q1 and Q2 It is going on all three So write all three

So obviously, you have only Q0 left. Q0, Q2 are going to 0. Q0 and blank. So there is no point in writing blank here. Because dead configuration does not exist in BFA. So you have to simply write Q0 here. Then what is going on 1? Q0, Q2. Q0, Q1 and dead.

Q0, Q1 and dead. So Q0, Q1 are as it is. That is how you have to expand. Now this has been expanded. Now we will expand straight. You just have to expand one by one in this way. So let's turn on your Q0, Q1, Q2. So let's expand Q0, Q1, Q2. That means you have three states. These three states. Q0, Q1, Q2. Look at 0. Q0, Q2, dead. Q0, Q2, dead. There is no need to write dead. Then what is going on 1? Q0, Q1, Q2, dead. Q0, Q1, Q2, dead. There is no point in writing dead.

So we have expanded it like this. In the case of 0 and 1. So this will also be straight. Next, let's go to Q0, Q2. We have already expanded Q0, Q2. Q0, Q1. We have already expanded Q0, Q1. So don't expand it again and again. What has already been done once, that is finished.

What will be the starting state? Same. The starting state which was in NFA, the same starting state will be in DFA. Final state? Final state was Q2 here. But what will be the final state in this? Final state will be the one in which Q2 is coming. Means all the states in which Q2 is coming, Q2 is coming in final.

I always had a burning desire to push myself further. So this is your DFA DFA that is Deterministic Finite Automata which accepts all binary strings in which second to last bit is 1 So when we designed DFA in the last video I said that when you design DFA you will feel that DFA designing is complex as compared to the NFA You can see in the diagram that which diagram is easy to design and which is difficult to design That's why it is difficult to understand DFA as compared to the NFA

And last point, whenever you design, whenever you convert, in the case of DFA, the number of states will be more. Because in the case of DFA, we don't have choices, we don't have data configuration. For every alphabet, for every binary string, we have to tell where it is going for 0 and 1.

There are various states in the case of DFA, which is not there in the case of NFA. So if I have N number of states in the case of NFA, then there is maximum 2 raised to power in DFA. It may be less, but maximum number of states possible is 2 raised to power.

If you design NFA, If the number of states is 3, then how many states are there in DFA? Maximum 2 raised to the power 3 which is 8. This is the maximum number. Here you can see that the number of states has increased from 3 to 4. But maximum can go up to 8. That is 2 raised to the power n.

Thank you.

Summary

**Overview**

This lecture explains how to design a Deterministic Finite Automaton (DFA) that accepts strings with specific counts of 'a's and 'b's. The focus is on four cases:

* Even number of 'a's and even number of 'b's
* Even number of 'a's and odd number of 'b's
* Odd number of 'a's and even number of 'b's
* Odd number of 'a's and odd number of 'b's

**DFA Design Process**

* The DFA consists of four states to track the combinations of even/odd counts
* Q0: Starting state (even 'a's, even 'b's) - accepts the empty string (epsilon)
* Q1: Represents odd 'a's, even 'b's
* Q2: Represents even 'a's, odd 'b's
* Q3: Represents odd 'a's, odd 'b's
* Transitions are defined between states for each input symbol ('a' or 'b')
* Self-loops are avoided as they would incorrectly accept strings with odd counts

**Final State Selection**

* To accept strings with even 'a's and even 'b's: Make Q0 the final state
* To accept strings with odd 'a's and even 'b's: Make Q1 the final state
* To accept strings with even 'a's and odd 'b's: Make Q2 the final state
* To accept strings with odd 'a's and odd 'b's: Make Q3 the final state

**Examples Validated**

* The DFA correctly accepts "ABBA" (two 'a's, two 'b's) when Q0 is final
* The DFA correctly rejects strings with odd counts when checking for even counts

**Educational Importance**

This DFA design pattern is noted as important for competitive exams and university-level coursework, as these questions are frequently tested.

Notes

Transcript

Next example on DFA designing is We have to design our DFA for the language L And that language accepts all strings W here denotes string But which strings? All strings Which has even number of A's and even number of B's Let me tell you that this question Even number of A, even number of B Apart from this, we will do 4 more questions Even number of A, odd number of B Odd number of B, even number of A And odd number of A and odd number of B These 4 questions will be covered in this video

It is very important from the point of view of competitive exams, even from the point of view of your college or university level. A lot of your questions are repeated. So guys, I am going to explain all the points in this video. So guys, quickly like the video and subscribe to the channel if you haven't done it yet.

And please press the bell button so that you get all the latest notifications. So let's start. First of all, we are focusing on even number of A and even number of B. And what is in between is AND. Both conditions should be true. In the case of AND, this should also be true.

This should also be true. When both are true, then the answer is true. So to fulfill the AND condition, first of all, we write the language. Language, which is a collection of... What if you could analyze the UX of any site with just a single click? Introducing UX Ray by Baymar.

So, first of all, we are going to create a starting state and an ending state. The reason for this is that I have not given any symbols in the starting state. Zero symbols. Neither did I give a, nor did I give b. And let me tell you that we are running on two alphabets.

We have two input symbols, a and b. So, neither did I give a, nor did I give b. So, I got it accepted in the starting state. But what is it accepting? It is accepting the epsilon. So, what was the minimum string? Epsilon. So, I neither got a accepted, nor did I give b an input.

So, neither a nor b are giving any input. So, by default, what do we have to accept? Now I have to do it for A and A. So let's suppose for A, we call it the next state. We call it Q0. We have Q1 as the next state. So we sent A to Q1. Now the first A comes. If one A comes, then what does one A mean? So we sent it to a new state. Let's say state is Q1.

So if the second A comes, then we want that if the second A comes, then it comes back. Coming back means that it comes to the final state. So that 2A means it should be accepted because it is uneven. So not 2, but 2A. If I make a self-loop here, what will happen? A, B, B, B, A. Is it accepted? Yes, this is also accepted. How? A, B, B, B, A. But this should not be accepted. Because it contains odd number of B. But this will be accepted. That means you don't have to make a self-loop in this.

You don't have to make a self-loop anywhere here or here. Because as soon as you make a self-loop, it is not necessary that the even condition will be accepted. The odd condition will also be accepted. So what we will do is, for B, we will send it to a new state.

Let's say state is Q3. So we sent it to Q3. And when I have, let's suppose, first B comes, first A comes, A, B comes, one B becomes one A. We want that if the second B comes, then for the second B, we send it back. So that when the fourth A comes, I mean, like this, A, B, B, A, the second string comes like this.

So look, A, B, B, A. Will it be set? Yes. So that's why we did it here for B and back for B. We are doing the same here in Q2. In Q2, let's say if I get B, A.

E, A, A, B A, B, B, A Double A followed by Double B Double B followed by Double A Even number of A, Even number of B

Oh, this DFA will accept. So see, my DFA has started for all states A and B. So this is how you have to design it. Now let me tell you the last point. Even number of A and even number of B is accepted by DFA if you make Q0, this state, final. But if you make Q1 final, if you make Q1 final, then making Q1 final means you didn't make this, you made this final.

If you made Q1 final, then what will it do? This odd number of A, even number of B. Odd A, even number of B. So you can check any swing in this way. But whatever you get in the question, Even A, Even B, then this one will come. Odd A, Even B, then this one will come. Odd B, here will come Even A and Odd B. So this one will come. If you take Odd A and Odd B, then this one will come.

So whatever question I have given you, check it carefully. You can decide the corresponding final stator for that. Thank you.

Summary

**Definition and Purpose**

* Regular expressions are a method to represent languages, similar to how Boolean expressions represent logical statements
* They represent regular languages - languages that can be accepted by finite automata
* Regular expressions provide a powerful, concise way to represent potentially infinite languages
* Used in compiler design and other applications in computer science

**Regular Languages and Automata**

* Regular languages are those accepted by finite automata like:
  + DFA (Deterministic Finite Automata)
  + NFA (Non-deterministic Finite Automata)
  + Epsilon NFA
  + Mealy Machine
  + Moore Machine

**Primitive Regular Expressions**

* Epsilon (ε) or lambda (λ): Represents the null string
* Phi (Φ): Represents the empty set (language with no strings)
* Single symbol (a, b, etc.): Represents a language with one string containing just that symbol

**Operations on Regular Expressions**

* Union (R₁ ∪ R₂ or R₁ + R₂): Combines strings from both expressions
* Concatenation (R₁R₂): Joins strings from both expressions
* Kleene closure (R\*): Represents zero or more repetitions of the expression
* Parentheses can be used for grouping

**Examples**

* A\* represents {ε, a, aa, aaa, ...} or all strings with zero or more 'a's
* A finite automaton with a loop on its start/final state represents the same language as A\*
* A ∪ B (or A + B) represents the language {a, b}

**Action Items**

* [ ] Practice writing regular expressions for given languages
* [ ] Study the relationship between finite automata and regular expressions

Notes

Transcript

And in this video, we are going to discuss the definition of regular expression and its introduction part. So guys, quickly like the video and subscribe to the channel if you haven't already. And please press the bell button so that you get all the latest notifications.

So let's start. First of all, why do we use regular expressions? What are they? Regular expression is actually a method to represent our language. If I want to represent a language, I use its regular expression. This is exactly the same as we use boolean expressions.

All the languages that accept finite automata are called regular languages. We have different kinds of machines like DFA, NFA, Epsilon MFA, Nele Machine, Muri Machine, which we have already discussed. All the languages that are accepted by these machines are called regular languages.

We use regular expressions to represent regular languages. We use it in compiler design as well. It is a very powerful method to represent a language. It is a collection of strings. Let's suppose I make a language. In this language, let's say first of all we have epsilon. Here we can also take epsilon because that symbol is actually a Greek symbol. Sometimes it gets confused. This is not log, so it is epsilon.

Take epsilon or take this symbol, lambda. So here we have, let's say, a, a, a, triple a, then further. Meaning, collection of a. So we have a to the power n, where n is greater than or equal to 0. So here, first of all, if we put the value of n at 0, then there is no a.

You can call it a null string. After that, single a, double a, triple a. Then it is going to... To represent this language, we can use a regular expression. Let's suppose we used A star. What is A star? You will understand further. Here, I am just telling you in the introductory part. It is difficult to represent this language in normal language.

Because in infinite language, there are infinite number of strings. So, we represented it in a simple way. A star. What does A star mean? In this star, you can put any number. You can put 0, 1, 2 or multiple times. So, this A star is representing collection of A.

Now, here A can be 0, 1 or multiple. Even if we talk about finite automata, if we don't want to make a finite automata, we can make it in a simple way, like this. Let's suppose we have a single input symbol, that is A. So, what did we make the first state?

Let's say this is a QNOTE state. We made it the final state. And we made the loop of A here. So, it accepts 0 number of A as 1, 2, 3. So, what can we write for the corresponding of finite automata? We can write regular expressions. What can we write for the corresponding of language?

We can write regular expressions. The next question is, Let R be a regular expression over alphabet sigma. If R is, the first important point is, this is again the epsilon. You can denote it like this. Epsilon is what? It is a regular expression denoting the set, this epsilon. So what is it saying here? If I have a regular expression.

Regular expression is epsilon. What does epsilon mean? It is a collection of null strings. What will be the corresponding language? Language will be this one. Collection of Strings What is the language here? Null String There is only one string in the language and that is Null String The regular expression is written like this and the corresponding language is written like this The next one is Phi The regular expression is Phi and it is the regular expression denoting the empty set This regular expression is denoting the empty set or you can say it is denoting the language in which there is zero number of strings There is no string The difference is clear

It is a collection of language without any symbol. Getting a shipping quote just got so much easier. Your competitive shipping rate. Lightning fast. A is a regular expression denoting the language. What is the language? It is simply You can make it for B or C. We call these three as primitive. What does primitive mean? What is the minimum we have? The minimum language we have is null, empty or single. A or single B. After this, we have union. Union of two regular expressions is also regular.

Let's suppose we have one regular expression R1 and one regular expression R2. If you take union of these two, what will happen? It will also be regular. This point is often used in exams. The three operators we use in this are union, regular and singular.

Concatenation and Clean Closer. Let's say we concatenate R1 and R2. R2 is a regular expression and there are two of them. The output will also be regular. Let's say I have R1, A, and union R2, let's say, B. I have two of them, A union B. What is the corresponding language?

Language in which I have only two strings, A and B. So, A union B. You can also write A plus B in between. A plus B. So, A and B is the language. Then we have Clean Closer. That is your epsilon. Then well, nothing is empty. Then we have a single a, single b. Now what can we do? What is the last point saying? If r is regular, then r in the parenthesis is also regular. And nothing else. Meaning, these are the seven points.

You can use these seven points recursively. You can use them repetitively. Through this, you can make multiple regular expressions. Other than this, there is nothing else. It's not like you can put any symbol in a regular expression. No. These are defined symbols.

These defined operators will have symbols. You have to use them. Then your regular expression will be... So what do we have to do? We have to write a regular expression corresponding to which? We have to write a regular expression according to the language given in the question. Or you can also use finite automata. You can also use a machine. You have to write a regular expression corresponding to that.

Summary

**Introduction**

This Geeksbashers video introduces regular expressions in Theory of Computation (TOC), explaining that this topic appears in competitive exams and university-level assessments. The fundamental concept presented is that all finite languages are regular, making it possible to write regular expressions for them.

**Key Concepts**

* Languages are collections of strings
* Finite languages are always regular
* Finite languages are always automatic
* Regular expressions can be written for any finite language
* The speaker references a previous video on formal definitions of regular expressions

**Regular Expression Examples**

* For languages with strings of length 1 (A, B): A + B
* For languages with strings of length 2 (AA, AB, BA, BB): AA + AB + BA + BB or (A+B)(A+B)
* For strings of length 3: expressions follow the same pattern
* For strings with at most 1 character: includes epsilon (empty string) and single characters
* For strings with no more than 2 Bs: various combinations like A, B, AB, BA, ABB, BAB, BBA

**Educational Context**

The video serves as instructional content for computer science students learning theory of computation, with emphasis on writing regular expressions for specific language constraints. The presenter encourages viewers to watch related videos for more comprehensive understanding of regular expressions.

Notes

Transcript

Which is an important topic of TOC And it is important because small questions can be asked Hello friends, I welcome you to Geeksbashers In this video, we are going to learn regular expressions Which is an important topic of TOC And it is important because small questions can be asked In your competitive exams, even in your college or university level exams So guys, quickly like the video Don't forget to subscribe to the channel And please press the bell button So that you get all the latest notifications So let's start with regular expressions First of all, let me tell you We have already discussed about language That it is a collection of strings

In this video, I am going to talk about Finite Languages. If a language is finite, it must be regular. If a language is finite, it must be automatic. It is also possible to write its regular expression. These three points are true. First, we will talk about regular expressions of finite languages.

How do we write its regular expression? Finite languages are always regular. If it is regular, I can write its regular expression easily.

There are many videos on how to use regular expressions. The link is given in the description box. You will get all the information on the formal definition of regular expressions. If you don't know about regular expressions, then how will you write regular expressions?

Do check that video so that you know the definition of regular expressions. Length 0 means the length of all the strings in the language. It has been edited to include proper punctuation. Now, when we write a regular expression for this, it is important to remember that whenever you write a regular expression for finite languages, you have to put a plus symbol in the middle. For example, A plus B. That's it. The regular expression for this is A plus B.

You just have to put a plus symbol in the middle. That is how we write the regular expression. So, what does A plus B mean? Either A or B. Either A or B. Because the length of both of them is the same. This one is also closed. And this one is also closed.

So this is also Length 1. Then Length 2. Meaning all the strings whose Length should be 2, what will be the language? AA, AB, BA, BB. This will be the chart. Nothing else can be made other than this. AA, AB, BA and BB. So see, the length of all this is 2. Now you have to write the regular expression for this. So how to write? a a plus a b plus b a plus b b. That's it. You have to use the plus symbol in between. If you want to simplify this a bit, then take out a common from here.

What will be a plus b? And if you take out b common from here, what will be a plus b? So you can write it like this. So in a way, what do you have? a plus b, a plus b. So if you multiply this, what will become? a a, a b, b a, b b. So it will become like this.

Write it like this. It has been edited to include proper punctuation. If you multiply A by B, you will get AAB. You can write ABA, ABB, BAA. This is a finite number. This is a regular expression of length 3. This is how you can denote it. At most 1. At most 1 means the length of the string should be at most 1. At most 1 means either 0 or length should be 1.

What is the meaning of 0 length which we have already written? It can be epsilon. It has been edited to include proper punctuation. Let's take another example. In a string, the number of bees should be not more than two bees and a number of bees. In your language, all the strings have a maximum value of B, not more than 2. And A should be of A only. So how can you make a note more than 2? Either you have neither B nor A. And if you want to make it like this, A can come alone, B can come alone.

A, B can come. B, A can come. And apart from this, we have to make a note more than 2 with B. So A, B, B can come. In this, one A came and two B came. B, A, B came. And apart from this, B, B, A came. So this is how we have to write the regular expressions for finite language. Thank you.

Thank you for watching.