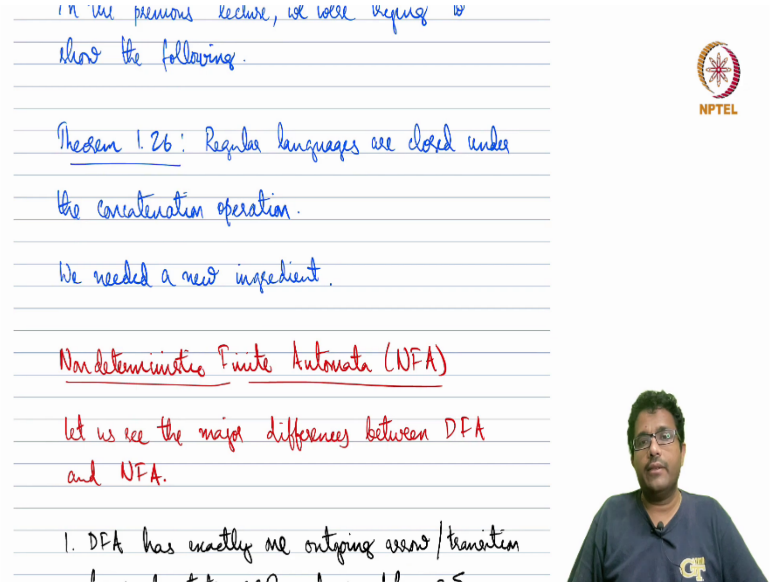
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Science and Engineering
Indian Institute of Technology Hyderabad
An Introduction to Non-Deterministic Finite Automata (NFA)

(Refer Slide Time: 00:16)



In the previous lecture, we were trying to show the following.

Theorem 1.26: Regular languages are closed under the concatenation operation.

We needed a new ingredient.

Nondeterministic Finite Automata (NFA)

Let us see the major differences between DFA and NFA.

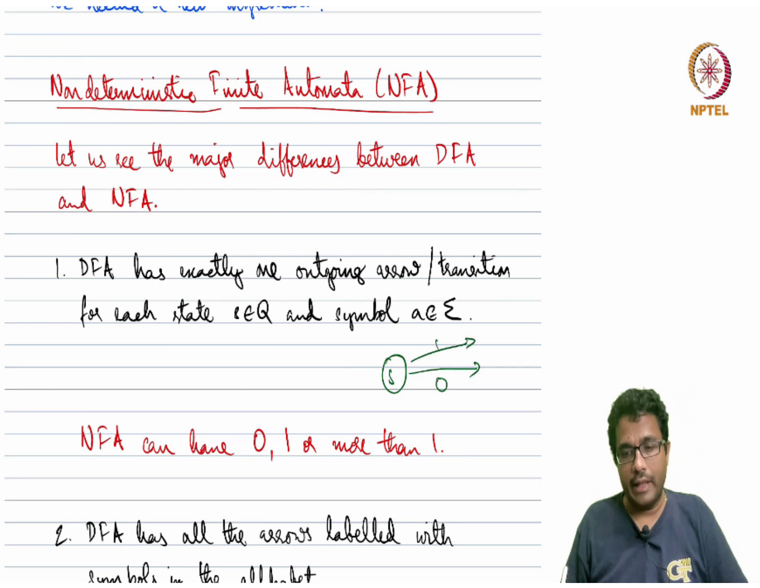
1. DFA has exactly one outgoing arrow/transition

Hello and welcome to week 2 of the course Theory of Computation and this is lecture 6. In week 1, we saw the introduction to the course, we saw deterministic finite automata DFA's, we saw that the languages recognised by DFA's are called regular languages. And then we saw that we saw some closure properties, we saw that regular languages are closed under complement, and then we saw that regular languages are closed under the union operation and then we said that there are operations called union concatenation star which are called regular operations and we said that regular languages are closed under all the regular operations.

So, the next thing to do was to show that regular languages are closed under the concatenation operation. And we realised it was not straightforward to show the same thing by using techniques similar to how we showed the closure under union. So, basically the model of DFA seemed limiting. So, towards this end, we said we wanted some other techniques and that is what we will see in this lecture.

So, we will see what is called non-deterministic finite automata. It is a model that is similar to DFA but seems to have a bit more power. It is called NFA in short, it is a non-deterministic, finite automata. So, definition of the same, like instead of deterministic, we have non-deterministic. So, let us see what are the major differences between DFAs and NFAs?

(Refer Slide Time: 02:01)



Nondeterministic Finite Automata (NFA)

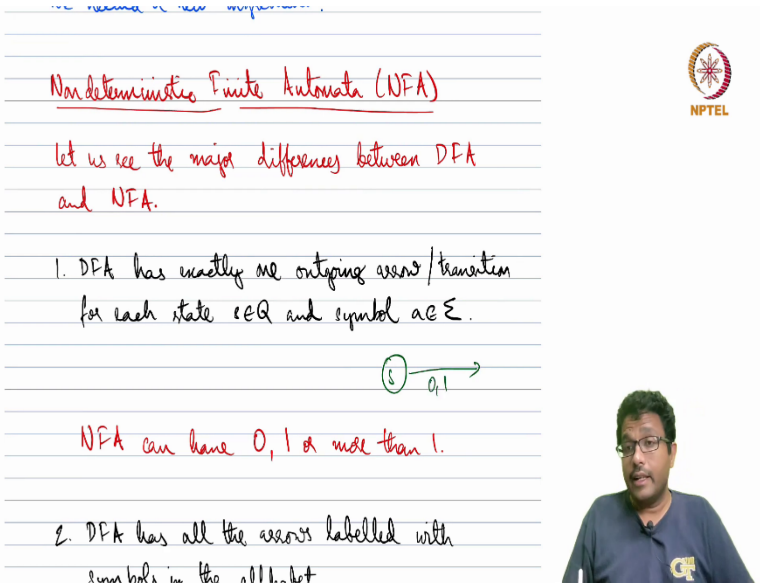
Let us see the major differences between DFA and NFA.

1. DFA has exactly one outgoing arrow/transition for each state $s \in Q$ and symbol $a \in \Sigma$.

$$\begin{tikzcd} s \arrow[r, "1"] \arrow[r, bend right, "0"'] & {} \end{tikzcd}$$

NFA can have 0, 1 or more than 1.

2. DFA has all the arrows labelled with symbols in the alphabet



Nondeterministic Finite Automata (NFA)

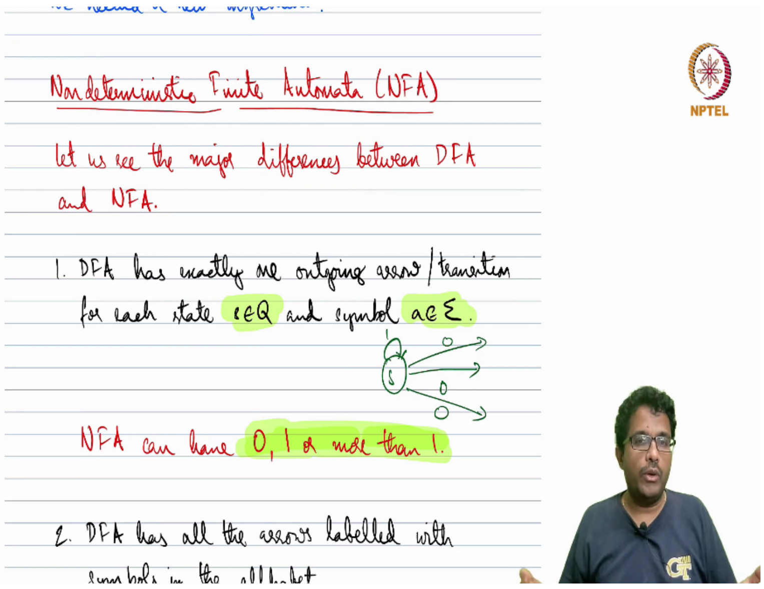
Let us see the major differences between DFA and NFA.

1. DFA has exactly one outgoing arrow/transition for each state $s \in Q$ and symbol $a \in \Sigma$.

$$\begin{tikzcd} s \arrow[r, "{0,1}"'] & {} \end{tikzcd}$$

NFA can have 0, 1 or more than 1.

2. DFA has all the arrows labelled with symbols in the alphabet



[illegible]

Nondeterministic Finite Automata (NFA)

let us see the major differences between DFA and NFA.

1. DFA has exactly one outgoing arrow / transition for each state $s \in Q$ and symbol $a \in \Sigma$.

$$\begin{tikzcd} s \arrow[loop above, "1"] \arrow[r, "0"] \arrow[r] \arrow[r, "0"] \arrow[r, "0"'] & {} \end{tikzcd}$$

NFA can have 0, 1 or more than 1.

2. DFA has all the arrows labelled with symbols in the alphabet

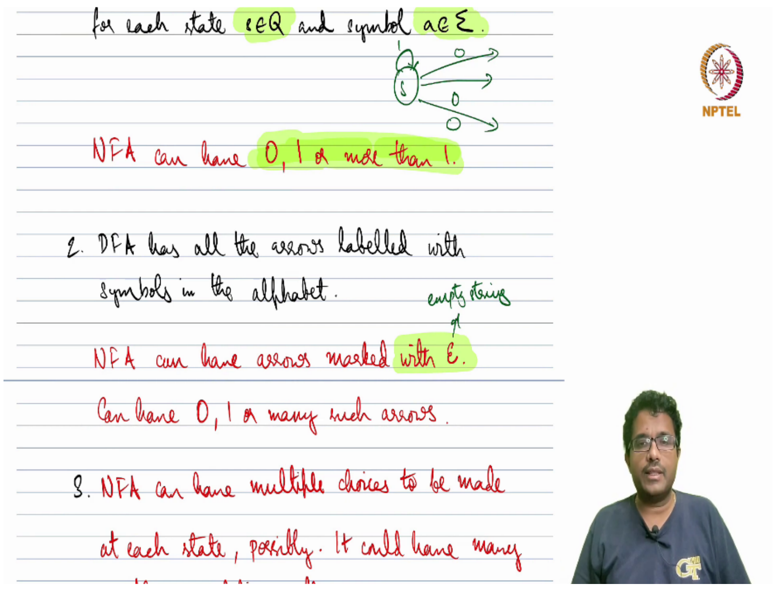
So, the major difference is that in a DFA, in a DFA, let us say it is a binary alphabet. So, if there is a state, S, and let us see, there are 2 symbols 0 and 1 binary alphabet, there is an outgoing arrow for 0 and an outgoing arrow for 1. There will be 2 such arrows, exactly 2 such arrows for each one for each symbol of the alphabet, it is possible that maybe both the arrows go to the same place. So, it could be something like this.

It is possible that one of the arrows are like one or both of the arrows are a self-loop, it is possible that both of those are self-loop, but the point is that there is one so if you are at s and you have 0, you know where to go next, if you are at s you see a 1, you know where to go next. So, for any state and for any symbol, you know what exactly is the next step to the next step to take. This is what makes a DFA deterministic. So, if you are at a certain state, and if you see a certain symbol, you know exactly what the next step is.

So, the current state and the symbol that you see, completely determines where you should go next. This is not the case in NFA's. So, in an NFA, the thing is that you could have multiple outgoing arrows with the same symbol. So, there could be 2 arrows or there will be 3 arrows with the symbol 0. There could be no arrows with the symbol 0 as well, there could also be 1.

So, the number of arrows with a specific symbol from a specific state. So, when I say arrows, I mean outgoing arrows, this could be 0, it could be 1, it could be more than 1, 0 1 or more than 1. So, the consequence is that sometimes there are multiple options available, sometimes there is no option available, sometimes there is only one option available. So, this is one of the main differences.

(Refer Slide Time: 04:10)



for each state $s \in Q$ and symbol $a \in \Sigma$.

NFA can have 0, 1 or more than 1.

2. DFA has all the arrows labelled with symbols in the alphabet.

empty string

NFA can have arrows marked with ϵ.

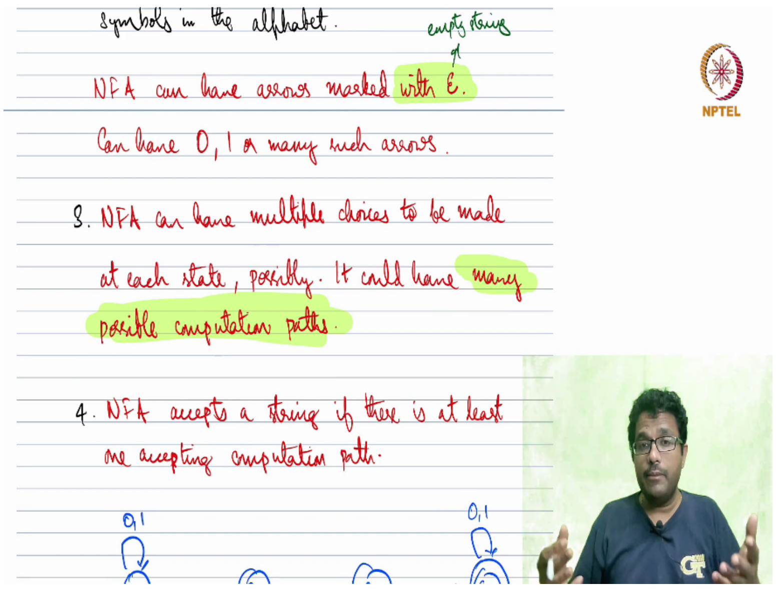
Can have 0, 1 or many such arrows.

3. NFA can have multiple choices to be made at each state, possibly. It could have many

The next thing is that in DFA's, all the arrows are labelled with the alphabet symbols. So, if the alphabet is binary, it is just the arrows are labelled 0 or 1. In the case of NFA's. You could also have arrows marked with ϵ, where ϵ is the empty string. So, this is the empty string. So, what this means is that if you recall, in a DFA, you read a certain symbol, and then you traverse an arrow and make a transition. That is what you did in a DFA in an NFA if there is an arrow marked with an empty string, you could make the transition. You could traverse the arrow marked with an empty string. But without reading any input symbols.

So, you could make the transition without reading any input symbol provided the transition, or the path is marked with an empty string. So, and not just that you can have 0, or 1 or many such arrows marked with empty string. So, in one state, you could have multiple arrows marked with epsilon. So, this may seem a bit confusing for now, but it is not all that confusing. We will soon see some examples.

(Refer Slide Time: 05:43)


symbols in the alphabet.

NFA can have arrows marked with $\overset{\text{empty string}}{\epsilon}$.

Can have 0, 1 or many such arrows.

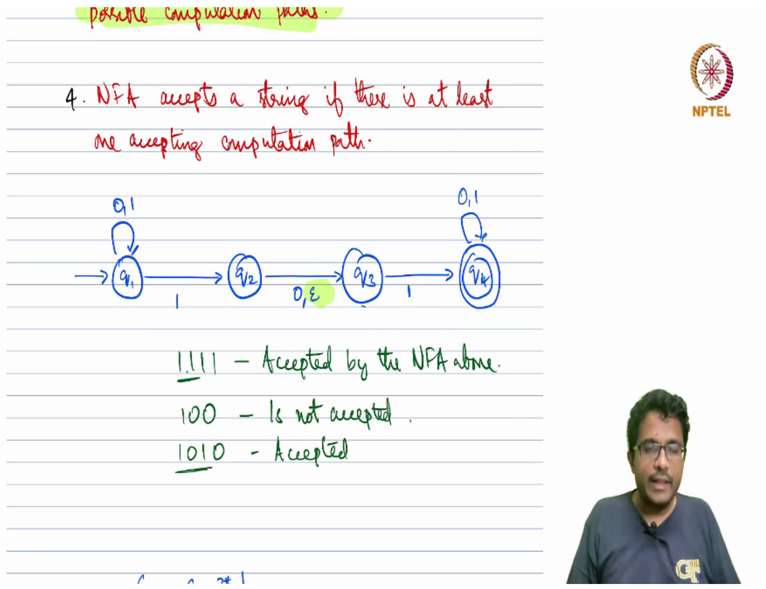
3. NFA can have multiple choices to be made at each state, possibly. It could have many possible computation paths.

4. NFA accepts a string if there is at least one accepting computation path.

So, as I already said, both of these things, so, one is that there could be multiple arrows with the same symbol, then there are these ϵ transitions, which means you can try, you can make a transition over a over an arrow, which is marked with ϵ, both of them, both of them result in some options for the NFA, like in a deterministic finite automata, you look at the next symbol, and then you know where to go next. And then you look at the next symbol after that, and then you go somewhere else. So, the process is completely deterministic, there is nothing for you to decide or determine or there is no choice or options available.

In the case of an NFA. The choice, there are choices, there are options available.This can lead to multiple possible computation paths. So, there could be many possible computation paths for the same input string. So, we will see an example, the same input string could have many possible computation paths. So let us see an example.

(Refer Slide Time: 06:51)



possible computation paths.

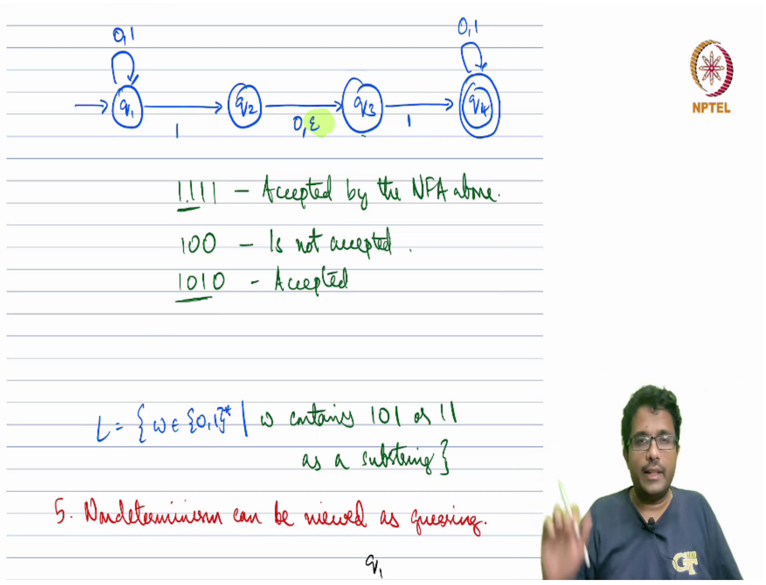
4. NFA accepts a string if there is at least one accepting computation path.

$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "0,1"] \arrow[r, "1"'] & q_2 \arrow[r, "0,\varepsilon"'] & q_3 \arrow[r, "1"'] & q_4 \arrow[loop above, "0,1"]
\end{tikzcd}$$

1.111 – Accepted by the NFA above.

100 – Is not accepted.

1010 – Accepted



$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "0,1"] \arrow[r, "1"'] & q_2 \arrow[r, "0,\varepsilon"'] & q_3 \arrow[r, "1"'] & q_4 \arrow[loop above, "0,1"]
\end{tikzcd}$$

1.111 – Accepted by the NFA above.

100 – Is not accepted.

1010 – Accepted

$L = \{ w \in \{0,1\}^* \mid w$ contains 101 or 11 as a substring $\}$

5. Nondeterminism can be viewed as guessing.

q_1

So, and one more final point is that in NFA, it accepts a string, if there is a way to process that string, such that once you complete processing that string, you end at an accepting state. So, in this particular NFA that I have drawn there is only one accepting state, which is q4. But you accept a string if out of the many possible options, there is at least one way to process a string ending at an accepting path. There could be other ways, which do not intersect at an accepting path. But if there is at least one way to end at an accepting path, you say that the string is accepted by the NFA.

So let us see what happens here. So let us consider some simple. Let us consider some simple strings. And let us see whether they are accepted or not. Let us consider the string 111, or let us say 1111. So, upon seeing the first one, you could remain at q1, you could just remain q1 throughout the first one, second one, third one fourth one. Which means it is not accepted, but then you have options upon seeing the first one you could remain you could go to q2 and then you make the transition there is an ϵ transition available here notice that there is an ϵ transition available.

So, from q2, you just make the ϵ transition, which means you do not read the input. So, you read the first symbol which was a 1 and then you came to q2, then you took the epsilon transition to q3. So, this arrow is marked with 0 as well as epsilon. So, and then, the second one is used to transition to q4. So, once you are at q4, there is only one option available. So, the third one keeps you at q4 this self-loop, and the fourth 1 also keeps you at q4.

So, then you end at q4. So, this is another way to process the same string ending at q4 which is an accepting state. So, therefore, this string is accepted by the NFA above. So, notice that this is indeed an NFA like it is; it has more flexibility. So, q1 has two outgoing arrows marked 1, one outgoing arrow marked 0, which is a self-loop, q2 has no outgoing arrow mark 1. And q2 further has an ϵ transition available, q3 has no outgoing arrow mark. So, if you reach q3, and the next symbol that you see is a 0 then you do not have a way to go.

So, let us consider another string. Let us see 100, suppose you take the transition 1, the first symbol takes you to q2, suppose you took that option, the first symbol could also you could remain at q1, but let us say you move to q2, the second 0, you move to q3. And now the third 0, there is nothing for you to do, because there is no outgoing arrow at q3, which is marked 0. So, then which means that you are stuck. There is also no ϵ transition, if there was an ϵ transition, you could take the epsilon transition and hope that the next state has a valid transaction available, but even that is not there.

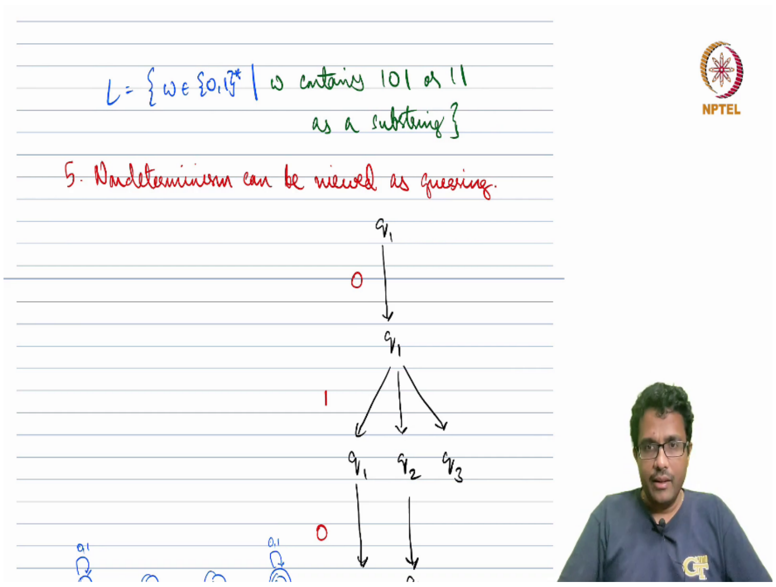
So, 100 does not get accepted in this manner. And if you look at it, it does not get accepted in any other manner as well. So, you can check this is not accepted. So, by that what I mean is, there is no way for this NFA to read 100 or process 100 such that it ends at the accepting state. So, the way we said so, if you are at a certain state and you do not have a valid outgoing arrow for the corresponding to the next symbol, that means that it is not a valid computation path available.

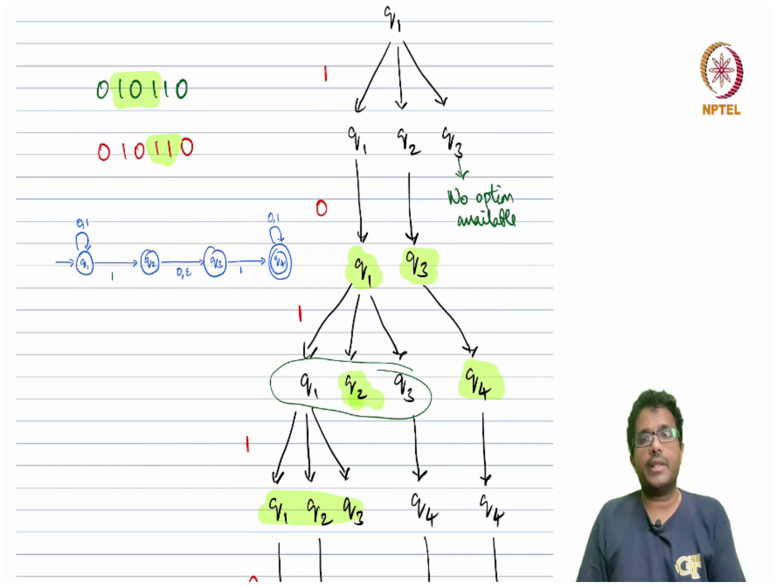
So, maybe, let us say some other string, let us say 1010. Of course, always, it is always an option to remain it q1 throughout, but then that is not good enough, because that does not take you to the accepting state. So, 1010 let us say that 1 takes you to q2, 0 takes you to q3, 1 takes you to q4 and then 0 keeps you at q4 the self-loop. So, this is accepted, because 101 you move to q4 and the last 0 you remain at q4. Notice that you this is the only way this could have been this could have been accepted if you chose to remain or if you chose to remain at q1, upon reading the first one then you have remained at q1 for the 0 as well, because that is the only outgoing 0 available, then if you move to q2 and then you take the 0 transition to q3, you cannot go to q4 because there is no further one.

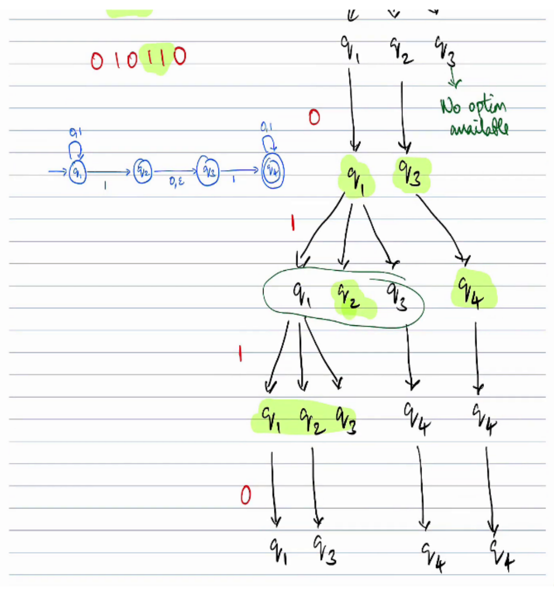
So, the only way to get accepted is to the first step, first 1 itself you move to q2 this the second symbol which is a 0 we move to q3 and third symbol which is a one you move to q4. So, if you notice, you can remain q1 throughout or you could and you can remain it q4 throughout the only way there are 2 possible ways to move from q1 to q4. One is that you have 101 somewhere in the middle of your string, which is what happened in the string here. There is 101 at the beginning. The other possibility is that instead of the 0 you could use the ϵ transition to move from q2 to q3 which means you should have 11 the string 11 somewhere in your string. Only then could you go from q1 to q4.

So, the language that is accepted here is the set of all binary strings where w contains 101 or 11 as a substring. So, unless 101 or 11 is there as a sub string which means in that order continuously without anything in between you are not going to get accepted. So you can try working out this and see that this is indeed the case. So, this is the idea of non-determinism. So, you see how even the same string presents us with multiple options and you accept if any of the valid computations lead to accept.

(Refer Slide Time: 14:25)

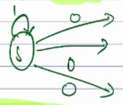

$L = \{ w \in \{0,1\}^* \mid w$ contains 101 or 11 as a substring$\}$
5. Nondeterminism can be viewed as guessing.
q_1
0
q_1
1
q_1 q_2 q_3
0





and NFA.

1. DFA has exactly one outgoing arrow / transition for each state $s \in Q$ and symbol $a \in \Sigma$.



NFA can have 0, 1 or more than 1.

2. DFA has all the arrows labelled with symbols in the alphabet.

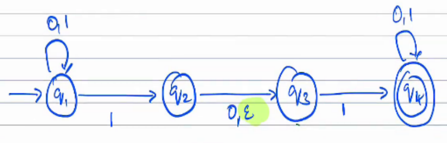
empty string

NFA can have arrows marked with ε.

Can have 0, 1 or many such arrows.

possible computation paths.

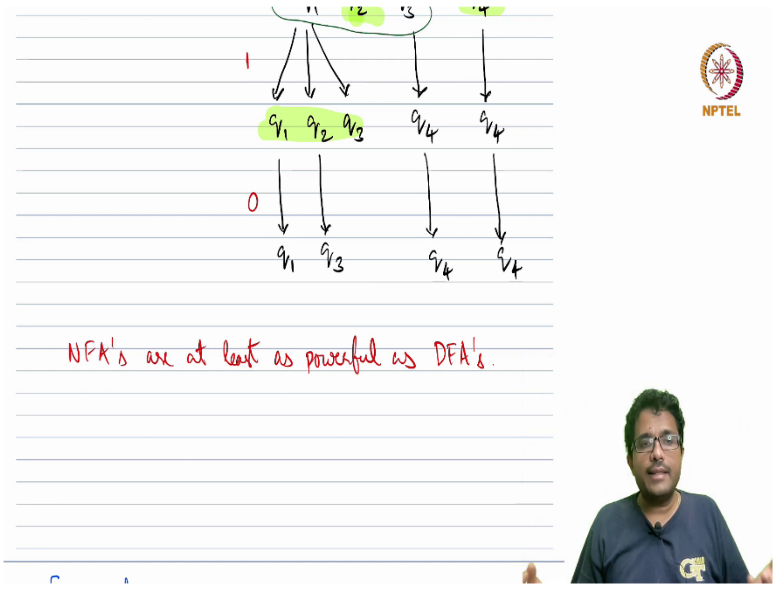
4. NFA accepts a string if there is at least one accepting computation path.



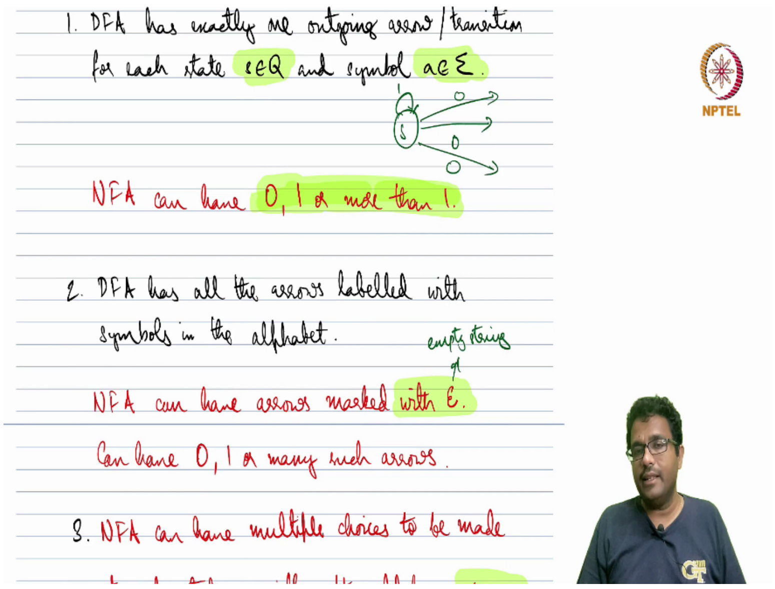
1111 — Accepted by the NFA above.

100 — Is not accepted.

1010 - Accepted



NFA's are at least as powerful as DFA's.



1. DFA has exactly one outgoing arrow / transition for each state $s \in Q$ and symbol $a \in \Sigma$.

NFA can have 0, 1 or more than 1.

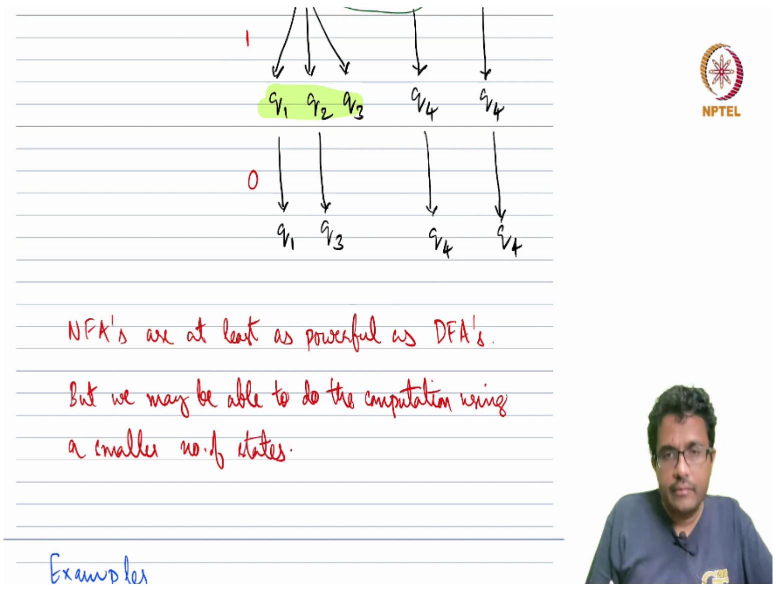
2. DFA has all the arrows labelled with symbols in the alphabet.

empty string

NFA can have arrows marked with ϵ.

Can have 0, 1 or many such arrows.

3. NFA can have multiple choices to be made



NFA's are at least as powerful as DFA's.

But we may be able to do the computation using a smaller no. of states.

Examples

Non-determinism can sometimes be thought of as guessing. So, there are multiple options available and the machine is able to guess the correct computation. So, now let us see another way to kind of see whether this computation or this machine is able to accept a certain string or not. So, the thing is that I have written here bit by bit the 010110 string is 010, I will write here 010110. So, the same NFA that is written over here I have just shrunk it and reproduce it here because I need space over the right side.

So, the first 0 keeps you at q1, because q1 has only one outgoing arrow with 0 on it, the second one the second symbol one1 there are 3 options you are at q1. So, if you were at q1 1 is to take the self-loop and remit q1. Other one is to take the outgoing arrow to q2. If you take the outgoing arrow and go to q2, you could also take the ϵ transition available to go to q3. So, what is it you remain in q1 or you go to q2 or you go to q2 and then take the ϵ transition. So, these are the 3 things that you can do upon reading the second one.

Now there are 3 possibilities after reading 01. You are at q1, q2 or q3, the third symbol is 0. If you are q1 then when you see a 0 you remain at q1 that is the only option. If you are q2 and then you see a 0 you can move to q3 using the q2 to q3 arrow. So I am referring to this third transition and the second transition here. If you are q3 and then see a 0, there is no outgoing arrow available. So, which means this part gets stuck there is no path here. No option available. No transition available upon seeing a 0 from q3.

So, q3 has only one outgoing arrow which is a 1. So now after reading 010, you could be at q1 or you could be at q3. These are the 2 places where you could be. Now, let us see what happens next. The next symbol is 1, if you were at q1 and see 1 we already saw it in the second step there are 3 possibilities either you take the self-loop the minute q1 or take the q1 to q2 arrow and then go to q2 or you take the q1 to q2 transition and then take the ϵ transition which means there are 3 possibilities which is q1, q2, or q3 that you saw in the second step as well.

If you are at q3 and see 1, then you reach q4, which is what we get here. So now after seeing 0101. All 4 states are possible q1, q2, q3, or q4. Now the next symbol is again 1, if you are q1 there are again 3 possibilities let q1, q2, q3. Upon seeing a 1, if you are q2 and see a 1 there is no possibility. So, this path gets stuck. If you are q3 and see a 1 you can go to q4 if you are q4 and see a one where there is a self-loop available. So, again q1, q2, q3, q4, all possibilities are available after seeing 01011.

But q4 there are 2 ways to reach q4. The others you can only reach one-way q1, q2, q3 and finally upon seeing a 0 from q1 you remain at q1, q2 you can go to q3, q3, there is no outgoing arrow, q4 you can remain at q4 using the self-loop. So, there are 4 possibilities. In fact, 3 possibilities q1, q3 and q4 upon seeing the string 010110. But the 3 possibilities, but then there are 2 ways to reach q4. One is to the 0 takes 0 keeps you at q1, then you move to q2 upon seeing 1, then you move to q3 upon seeing a 0, then you move to q4 upon seeing 1.

So, what happens in the first case is that this 0, this 101 takes you to q4 and then you remain at q4. The second case is when 010110 the second case you remain at q1 till you read 010. Then you move to q3 upon seeing the 4th symbol that is a 1. So, q1 to q2 and then epsilon transition and then you move to q4 upon seeing the fifth symbol which is a 1. So, this 11 takes you to the accepting state. So, there are 2 ways to read the string and accept. So again, this tree, we call it the computation tree. It captures the 2 different ways the string could be processed or not many different ways this string could be processed. In fact, there are 4 valid ways of completely processing it and two of them need to accept. So, which means this string will be accepted. So, you do not need to you just need one to be accepted.

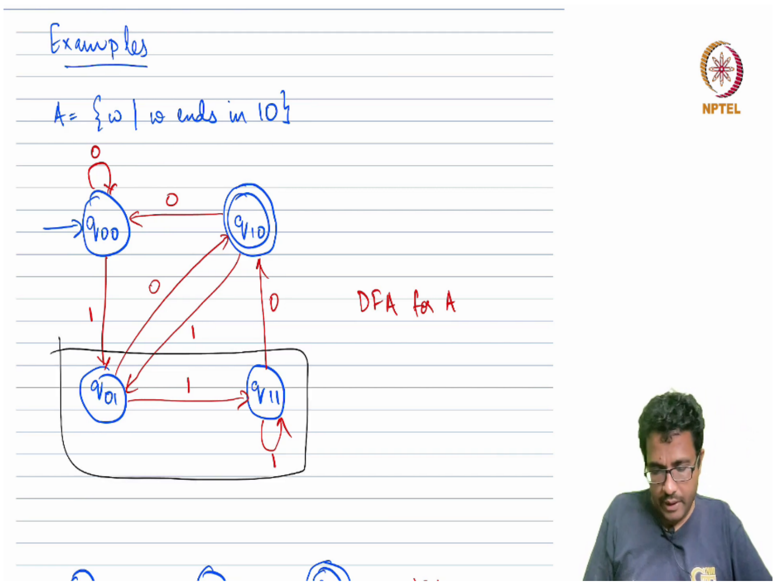
So 010110 is accepted by this NFA and this is consistent with what we wrote above. It contains 010 or 11 as a substitute in fact contains both. So, this is what NFA’s are. So, and this computation tree captures everything possible. So, the thing is, it must be clear now that NFA’s are at least as powerful as DFA’s, one is that you could have multiple outgoing arrows or 0 outgoing arrows. And then the second thing is that there could be ϵ transitions, empty which transition on an empty string. And the third thing that we said is if you accept if there is at least one way to accept.

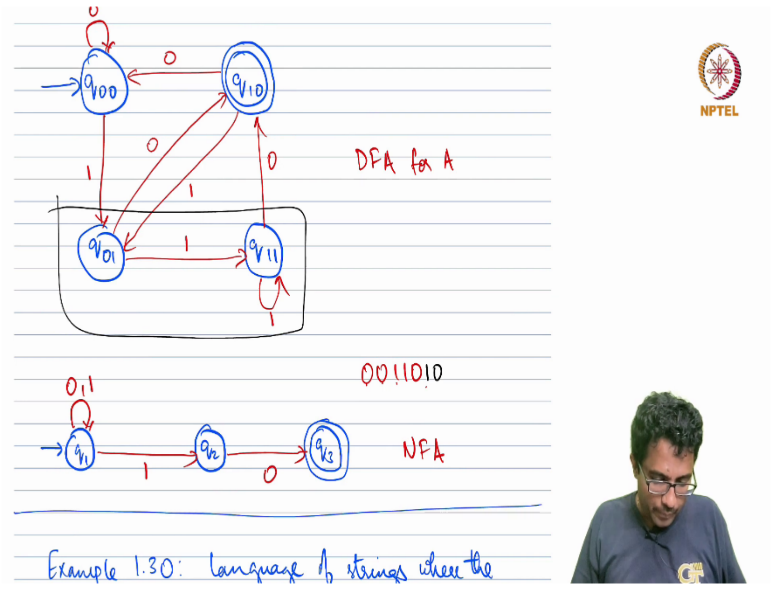
So, one point is that for NFA's, you can do anything that you can do with a DFA you can do with an NFA because the DFA itself is an NFA. So, recall regarding point one, so we already say that 01 or more than one is possible. So, one is possibility. So, DFA is an NFA, where every time there is exactly one outgoing arrow. And the ϵ transition we just refuse to use in an NFA then we get a DFA. So, anything that can be done with the DFA can be done with the NFA. So, NFA’s, or at least as powerful as DFA is because DFA itself is an NFA. And we may be able to do the computation using a smaller number of states.

So, a natural question that our NFA is more powerful. Is there something that an NFA can do, but a DFA cannot do? The answer is, the answer is that no, everything that an NFA can do, you can write a DFA for the same thing, just that you can perhaps get a simpler NFA for it. So, what that means is that NFA's, even though it seems to be more powerful, the class of languages recognised by NFA's, are exactly the same as the class of languages recognised by DFAs.

So, we will soon see that we already said that anything that you can do with a DFA can be done with an NFA, which is straightforward, you do not need a proof. But we will also see that anything that can be done with an NFA, you can write a DFA for it. So, this may be a bigger DFA, more involved DFA, but it does not matter. It is a finite state machine. So, it will be a DFA. So, we will see later that NFAs and DFAs are kind of equivalent in power in the sense that if you do not bother about the number of states etcetera, anything that an NFA can do a DFA can also do and vice versa. So, the class of languages recognised by NFAs are also the same as the regular languages. So let us see some examples.

(Refer Slide Time: 24:29)





So, one example is a set of all binary strings that end in 10. So, one way to do that using a DFA is to keep track of the last 2 symbols that you saw. So, we have 4 states q 00100111, and accepting status 10, so where you keep track of the last 2 symbols. So, for instance, if the last 2 symbols that you saw was 00 and, if you see a 1, if you see a 1, then the last, this 0 becomes a penultimate symbol seen in the last symbol that you saw would be a 1. If you see a 0 basically it is a self-loop. From 01, if you see a 1 it is like it is it, basically you will see two 1s at the end.

If you see a 1 here, it is a self-loop. And if you see a 0 from here, basically it is like, no, sorry. If you see a 0 from here, it is like this. And if you see a 0 from here, it is like, again, the state 0 from q 11, meaning the last 2 symbols that you saw was 11, if the next symbol is 0, so it is like 0110. So, the last few symbols are 10. And finally, over here, if the last symbol that you saw was a 0, you go back to q0. And the last symbol that you see is a 1, you come to 01, so it is like 101. So, which means the last few symbols are 01.

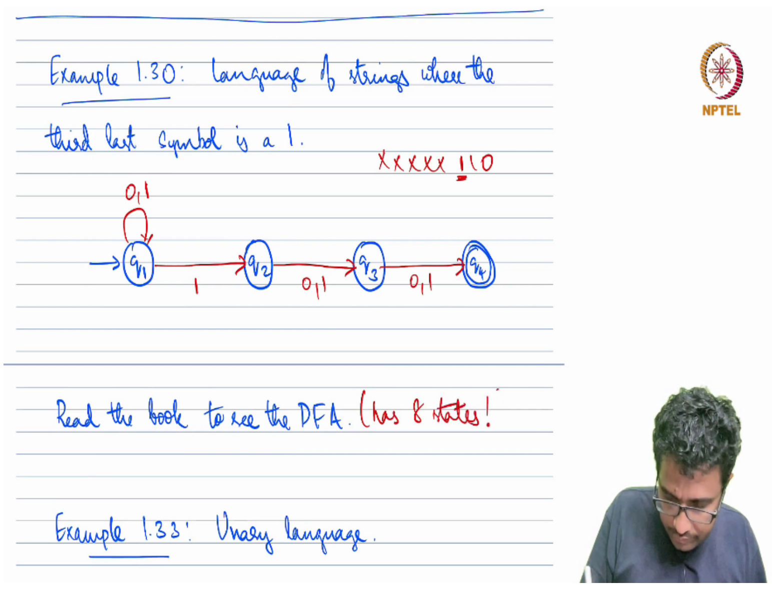
So, this is one way to keep track of the last 2 symbols. And this recognises this accepts all the strings, which ended 1,0. In fact, this is not the smallest DFA, this is just simpler to describe. In fact, you can, you can do something slightly smarter, you can combine these 2 states, sorry, you can combine these 2 states, I think. And basically, you are going to get a 3 state DFA. But an NFA turns out to be really easy, which is what I wanted to say. Basically, you want the last 2 symbols to be 1, 0. So, you transition when you see a 1, you transition when you see a 0. You do not put any other options for q2 and q3. The only other option that you give is for q1, you have a self-loop, when you are at q1.

So, the only way to come to an end at q3 and get accepted, if you notice, is to remain stuck at q1 till you see the last 2 symbols. So, it is the way to accept this thing 00110, is to 0 you remain at q1, 0 you remain at q1, 1. So, 1 you remain at q1 and the second the 4th one the 4th symbol 1 you move to q2 and the final symbol 0 you move to q3. So, suppose you so this is the way to end at q3. Suppose you move to q2 with the third symbol 1, then you cannot proceed. This is not a valid computation path because you are at q2 and then you have another one coming, then you do not know what to do next.

So just to give another example, suppose this string was 0011010. Now suppose you 001 you remain at q1, 001, the 4th symbol 1 you move to q2, and the fifth symbol, 0, you move to q3. Now, again, you are stuck because there are 2 more symbols. But there is no outgoing arrow out of q3. So, the only way to end at q3 in a valid legal way is to move to q2 in the penultimate symbol, and move to q3 in the last symbol. Because if you come here early, and then there are no more options available, that means it is not a valid computation, you have to end, you have to read all the symbols.

If you read part of the string and end in an accepting state, but there are still there are strings, the part of the strings remaining, that is not a valid computation, you have to read the entire string and end at an accepting state. So, the way to accept this string, this string right now is for all the red symbols, you remain it q1, the 6th symbol 1, you move to q2 and the last symbol 0 you move to your q3. So you see that this is an NFA and this seems to be rather simple.

(Refer Slide Time: 29:56)



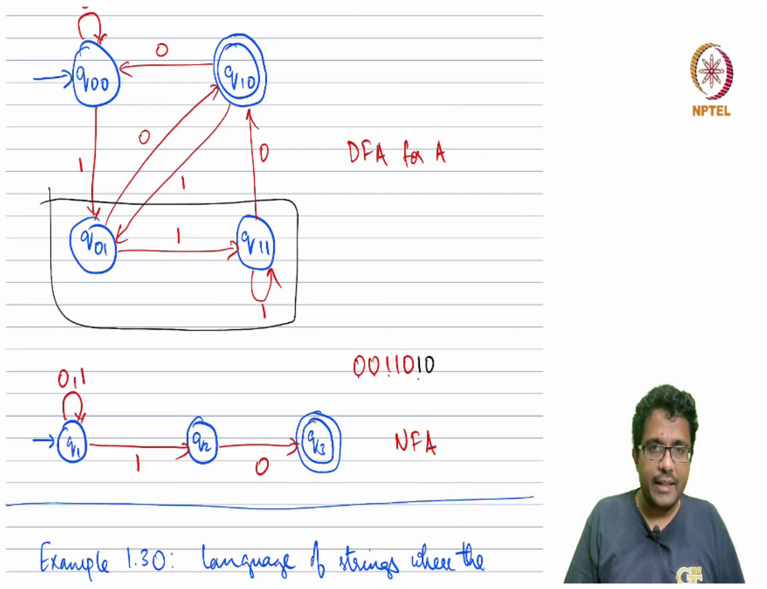
Example 1.30: language of strings where the third last symbol is a 1.

XXXXX 110

$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "0,1"] \arrow[r, "1"'] & q_2 \arrow[r, "0,1"'] & q_3 \arrow[r, "0,1"'] & \boxed{q_4}
\end{tikzcd}$$

Read the book to see the DFA. (has 8 states!

Example 1.33: Unary language.



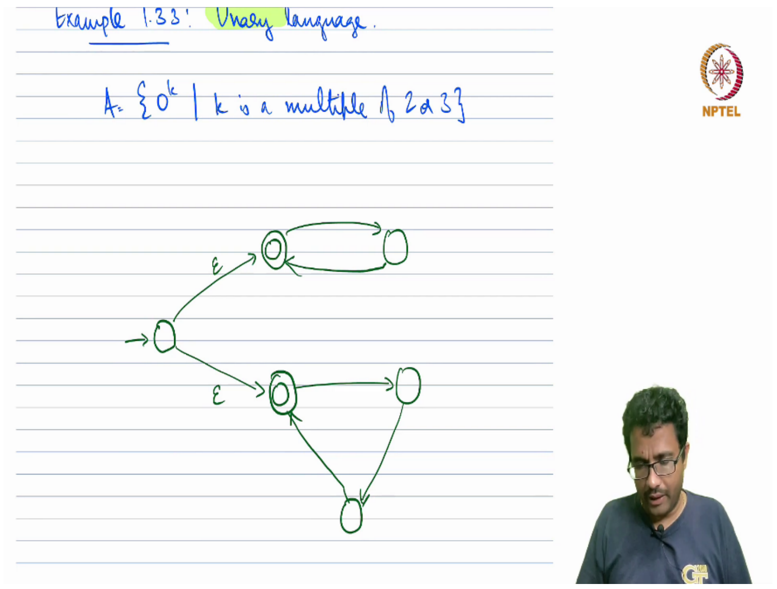
Finding another NFA another NFA example is, we want to accept all the, let us say, again binary alphabet, you want to accept all the strings, where the third last symbol is the 1. Again, the way to do it is, you remain stuck at the first state and you should be able to move from q1 to q2 to q3 to q4 over the last 3 symbols. The last 3 symbols should take you to q4 and you want to ensure that the third last symbol is a 1. So, the third last symbol transition, q1 to q2 should happen at a 1. The second last symbol you do not care about, it could be 0 or 1. And the last symbol is also okay to be 0 or 1.

So, now the only way to end the processing of the string in the valid way, is to start making this transition q1 to q2, q2 to q3 and q3 to q4 over the last 3 symbols of the strings. But, then if you have to make this transitions over the last 3 symbols, so some kind of let us say xxxxx, let us say 010, you cannot, this is not valid because this 0 has to take you from q1 to q2, but then that is not allowed. The only thing that can take you from q1 to q2 is a 1. So, this will not be accepted. So, this will be accepted, because this one, underline 1 will take you from q1 to q2. The next one will take; the 2 symbols will take you from q2 to q3 and q3 to q4.

So, if you read the book, this question is there in the sipser textbook, there is also the DFA that is illustrated and this DFA has 8 states and that is the smallest DFA that can recognise the same language. So, you read the book, please read the book to see the DFA and that is, perhaps make you appreciate the fact, that NFA is like very simply, very easy to understand only like you just see it is almost a straight line, except for the self-loop. The DFA has 8 states with all kinds of transitions going across.

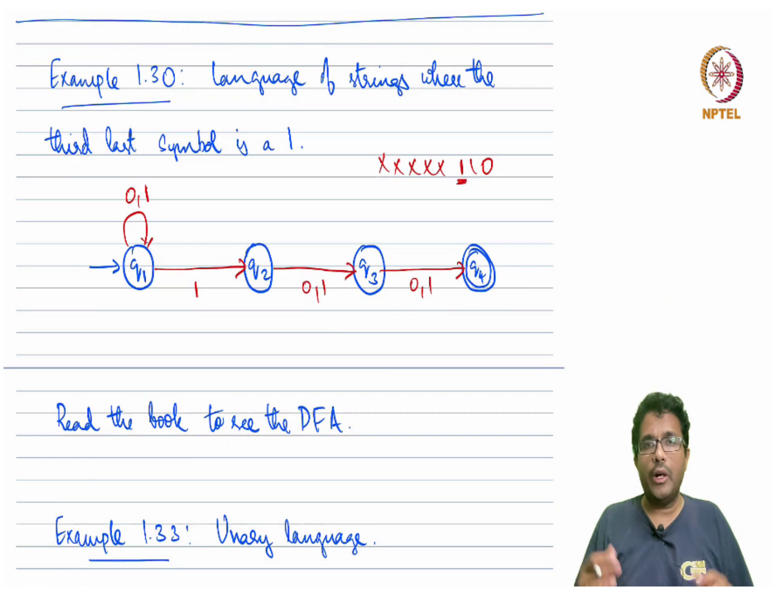
Something similar to this idea but extended to 8 states, because you have to keep track of all the last 3 symbols that you saw. So, this has 8 states, and you cannot simplify it further. So, you have a look.

(Refer Slide Time: 32:42)



Example 1.33: Unary language.

$A = \{0^k \mid k \text{ is a multiple of } 2 \text{ or } 3\}$



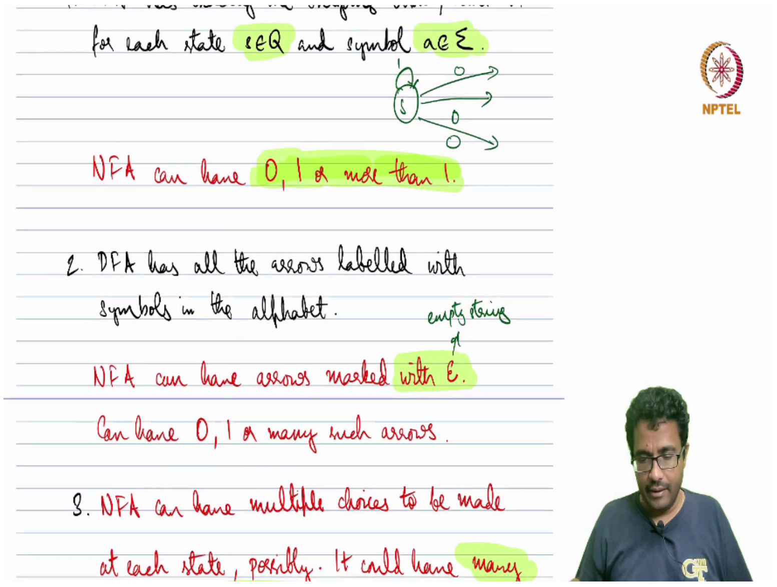
Example 1.30: language of strings where the third last symbol is a 1.

xxxxx110

$$\begin{tikzcd} \arrow[r] & q_1 \arrow[loop above, "0,1"] \arrow[r, "1"'] & q_2 \arrow[r, "0,1"'] & q_3 \arrow[r, "0,1"'] & q_4 \end{tikzcd}$$

Read the book to see the DFA.

Example 1.33: Unary language.



for each state $q \in Q$ and symbol $a \in \Sigma$.

NFA can have 0, 1 or more than 1.

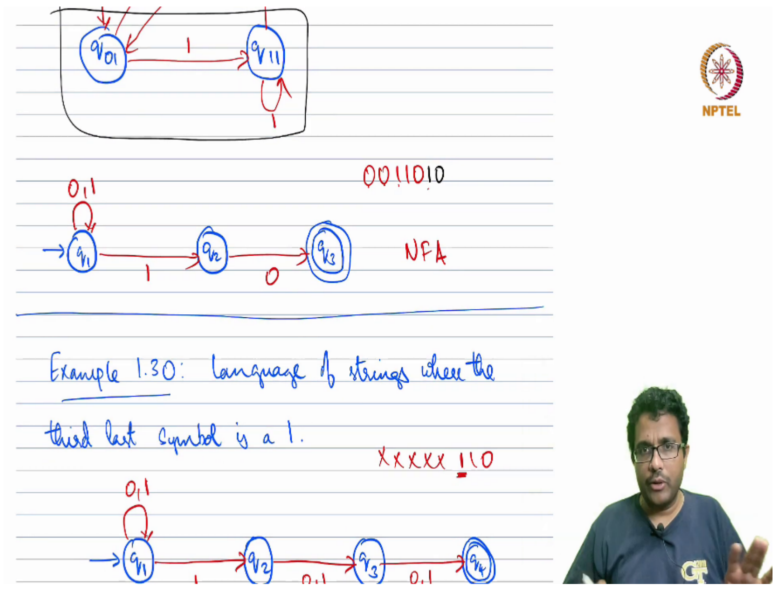
2. DFA has all the arrows labelled with symbols in the alphabet.

empty string

NFA can have arrows marked with ε.

Can have 0, 1 or many such arrows.

3. NFA can have multiple choices to be made at each state, possibly. It could have many



$$\begin{tikzcd} q_{01} \arrow[r, "1"] & q_{11} \arrow[loop below, "1"] \end{tikzcd}$$

0011010

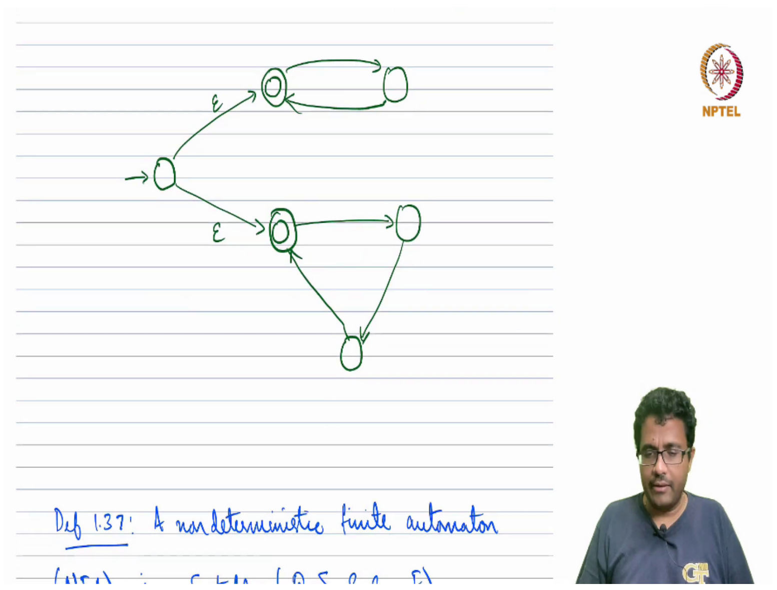
$$\begin{tikzcd} \arrow[r] & q_1 \arrow[loop above, "0,1"] \arrow[r, "1"] & q_2 \arrow[r, "0"] & q_3 \end{tikzcd}$$

NFA

Example 1.30: Language of strings where the third last symbol is a 1.

xxxxx110

$$\begin{tikzcd} \arrow[r] & q_1 \arrow[loop above, "0,1"] \arrow[r, "1"] & q_2 \arrow[r, "0,1"] & q_3 \arrow[r, "0,1"] & q_4 \end{tikzcd}$$



Def 1.37: A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

And finally, I have one more language which is unary language, unary means it is a language over one symbol. So, in this case the symbol is 0. So, I accept the, so which means any string is entirely composed of 0’s and I accept the string if the length of the string. So, the only parameter is the length, because all the symbols are 0’s. I accept the strings if the length is the multiple of 2 or 3. Again you can make DFA for it, it is not that hard, but the NFA is very simple, I am not bothering to explain but you can have a look.

So, just to summarize, we saw what a NFA in a high level is in an informal sense. We did not formally define it yet and we saw examples of what gets accepted and what does not get accepted. So, now we know the rules, or even though not formally so, we at least know the rules in an informal sense. We understand the rules. So, the next step is to define the NFA formally, which we will do in the next lecture video.