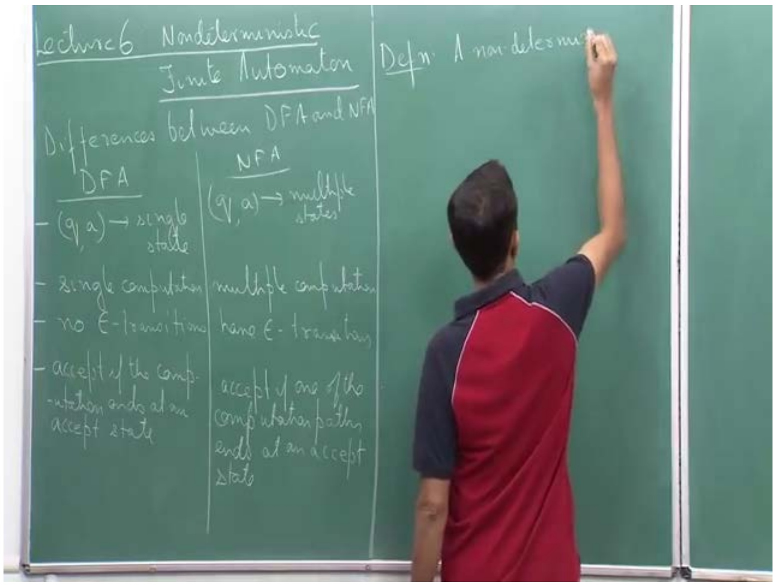
Theory of Computation
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 06
NFA, Definition and examples

Welcome everybody, this is the 6th lecture. And today, we will talk about Nondeterministic Finite Automaton.

(Refer Slide Time: 00:18)



Lecture 6 Nondeterministic Finite Automaton

Differences between DFA and NFA

DFA	NFA
- (q, a) → single state	(q, a) → multiple states
- single computation	multiple computation
- no ε-transitions	have ε-transitions
- accept if the computation ends at an accept state	accept if one of the computation paths ends at an accept state

Defn. A non-determinis

So, I will begin by reviewing the difference between a deterministic finite automaton and a nondeterministic finite automaton. Then I will give the formal definition of what a nondeterministic finite automaton is and what does it mean, what is the definition of computation by nondeterministic finite automaton, and I will end with few examples.

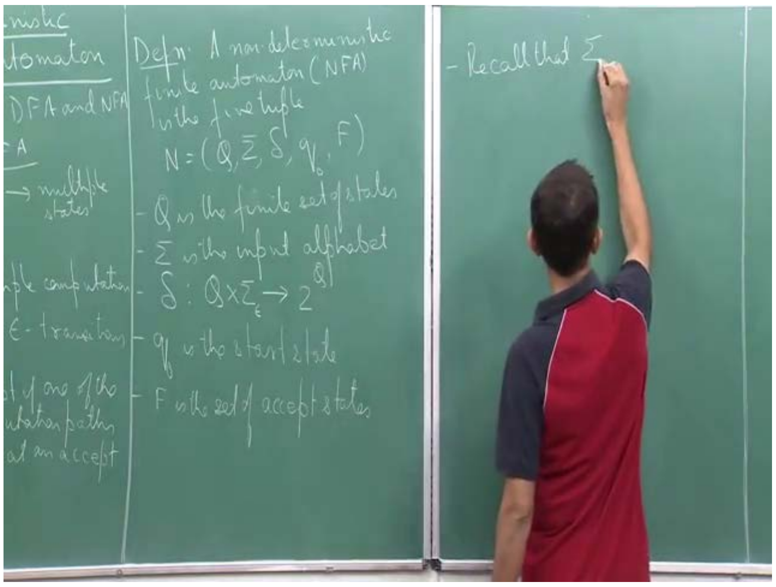
Let us look at some difference. So, we have DFA, and we have NFA. The first is in a DFA, from a state comma symbol pair, so from let say a state comma symbol pair, we go to single state always. So, this is the most important feature. Whereas in the case of an NFA, from a state comma symbol pair, we can go to multiple states - the second point is, in a DFA, we have a single computation that is happening. There is so one way of thinking of it is that, so if you think of the clone model that I talked about last time, in the case of a DFA, there is only one person. This person who is actually walking on this automaton is more like a person from our planet. So, it does not create any clones of himself or herself.

At every point of the computation, it read reads the next input bit and it moves to the next state. So, there is the single computation that is happening. In the case of an NFA, we have multiple computations that are happening simultaneously. So, because of this fact that from one state, we can go to multiple states, and each of those computations can proceeds independent of one another in a simultaneous manner, we have multiple different computations.

The third point is that of epsilon transition. In a DFA, we do not have any epsilon transition. By definition, here we have epsilon transitions. The next point is that of acceptance in the case of a DFA, the definition of acceptance is pretty simple because there is only computation path. At the end, either you are at an accept state or you are not at an accept state. So, you are of if you are at an accept state, then you accept; otherwise you do not. So, here accept, if the computation ends at an accept state.

In the case of a nondeterministic finite automaton, the definition is different; so here we accept if one of the computation paths ends up at an accept state. So, these are the fundamental difference. So, just the compliment fact is that when do we reject an input, again in the case of a DFA, we reject an input, if the unique computation path ends up at a non accept state; here we will reject the input, if none of the computation paths end up at an accept state. So, basically all computation path ends up at a non-accept state.

(Refer Slide Time: 06:13)


Defn: A non-deterministic finite automaton (NFA) is the five tuple $N = (Q, \Sigma, \delta, q_0, F)$

- Q is the finite set of states
- Σ is the input alphabet
- $\delta : Q \times \Sigma_\epsilon \rightarrow 2^Q$
- q_0 is the start state
- F is the set of accept states

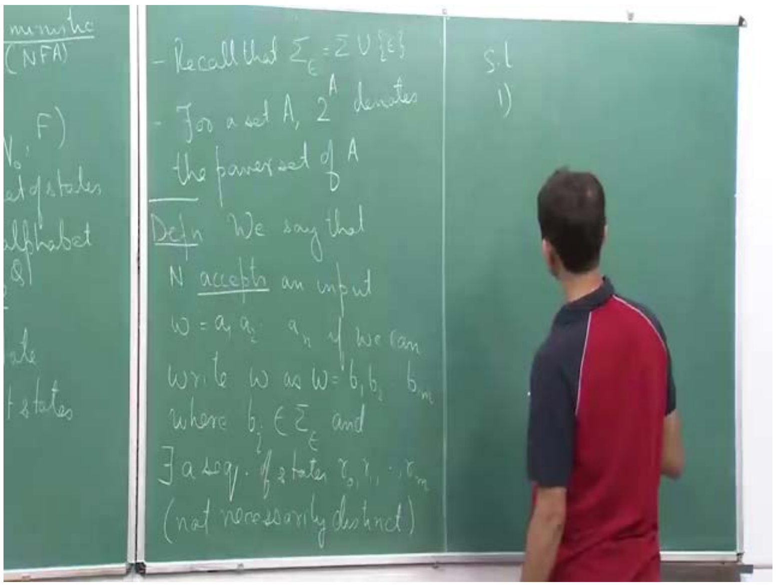
- Recall that Σ_ϵ

Now let us formally define what an NFA is. So, a nondeterministic finite automaton which in short we call NFA is the 5 tuple, N equals Q, sigma, delta, q naught comma F. Once again, we have a 5 tuple; and basically except for the transition function which is slightly different in the case of an NFA, the other symbols are basically, they denote the same thing. So, where Q is the finite set of states; sigma is the input alphabet; I will just come to the transition function in a moment; q naught is the start state; and F is the set of accept states.

So, what is the transition function, how is the transition function defined here. Let us recall how transition happens in the case of an NFA. In the case of a DFA, from one state on an input symbol, we go to a unique state. Therefore, it was a map from Q cross sigma to the set Q itself, because we went to a single state. In the case of an NFA, from a state comma symbol pair, we can go to multiple states and that can include the empty set as well.

In the case of an NFA, we have transitions which go from Q cross sigma. So, it is a function from Q cross sigma to the power set of Q; basically the set of all subsets of Q, so this we denote as 2 to the power Q. And this is not it, so this is not complete because also in the case of an NFA, we have epsilon transitions. So, we need to accommodate for that as well. So, from a state Q, we can see a symbol which is from the input alphabet or we can also see epsilon transition, instead of sigma we will write it as sigma epsilon.

(Refer Slide Time: 10:25)



- Recall that $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$
- For a set A, 2^A denotes the powerset of A

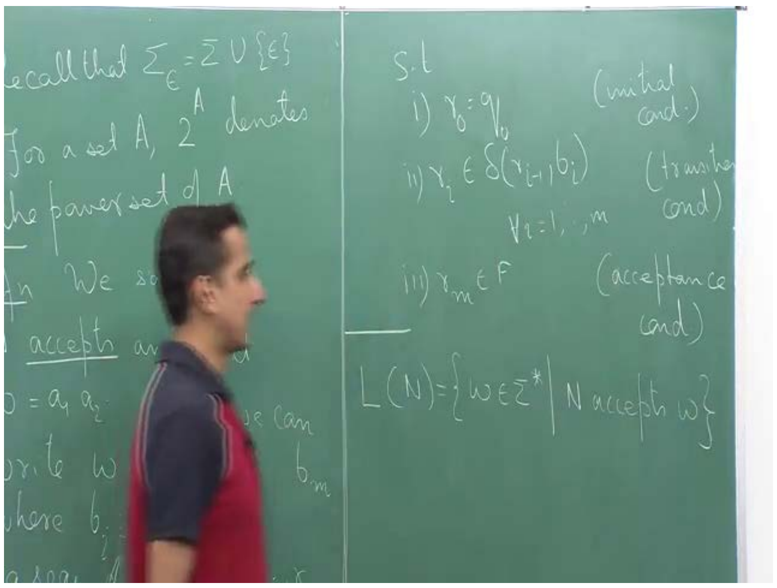
Defn We say that N accepts an input $w = a_1 a_2 \ldots a_n$ if we can write w as $w = b_1 b_2 \ldots b_m$ where $b_i \in \Sigma_\epsilon$ and $\exists$ a seq. of states $r_0, r_1, \ldots, r_m$ (not necessarily distinct)

(NFA)
F)
et of states
alphabet
ate
f states
S.t.
1)

Let me just for the sake of completeness, let me write this. So, recall that sigma epsilon is basically sigma together with the symbol epsilon and for a set A, 2 raise to the power A denotes the power set of A. So, this is what is the definition of an NFA. Now, how do we define acceptance, let say we have this NFA. So, this was the definition of a nondeterministic finite automaton. We say that n accepts an input w, which is equal to let say a 1, a 2 up to a n.

Let say - we have a string w, which has n symbols in it, a 1 through n; when do we say that nondeterministic finite automaton, accepts w. So, we say that n accepts a 1 w, if we can write w as w equals b 1, b 2 up to some b m, where b i is belong to sigma epsilon, and their exist a sequence of states r 0, r 1 up to r m, not necessarily distinct such that.

(Refer Slide Time: 14:02)


ecall that $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$
For a set A, 2^A denotes
he powerset of A
An We sa
accepts a
$= a_1 a_2$
rite w
here b_i
S.t
i) $r_0 = q_0$ (initial cond.)
ii) $r_i \in \delta(r_{i-1}, b_i)$ $\forall i = 1, \ldots, m$ (transition cond)
iii) $r_m \in F$ (acceptance cond)
$L(N) = \{w \in \Sigma^* \mid N \text{ accepts } w\}$

So, before I complete this, let us try to paths what we have written so far. So, suppose I have a string w, which is a 1 through a n. We say that our nondeterministic automaton accepts w, if I can write w as a sequence of symbols b 1 through b m, where this b i comes from sigma epsilon. So, some of this b i(s) can also be the epsilon symbol. So, in other words, if I just concatenate all these symbols, I will get w. So, I have some extra epsilons over here.

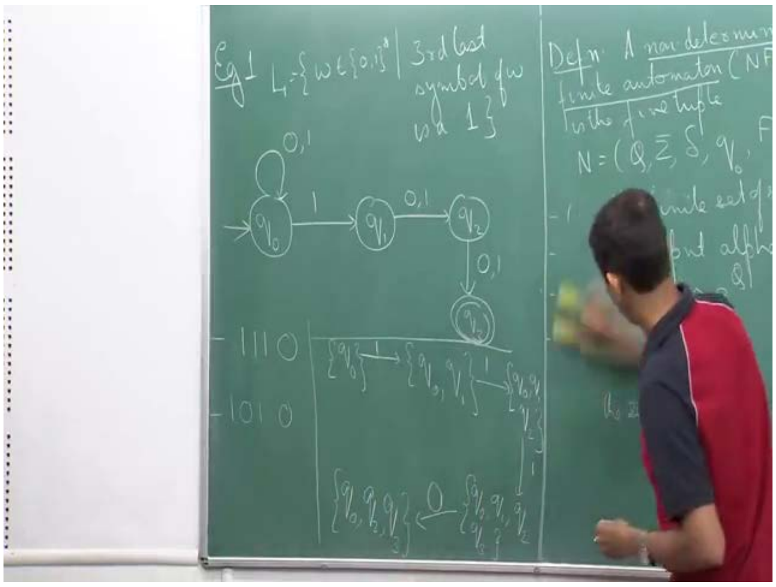
And there is a sequence of states r 0 up to r m. So, one point that needs to be noted here is that note that here I have n and here I have m, and of course, n is a number that is greater than or equal to n because I have this additional epsilon symbols plugged in between. So, and we have this sequence of states r 0 to r m and these states are not necessarily distinct, so some of this states can be the same state.

Such that we first we have to write the initial condition that is r 0 equals q 0; so this is the initial condition. Secondly, what is the transition condition here, how can we go from a state r i to a state r i plus 1. So, r i is contained in delta of r i minus 1 comma b i, for all i going from 1 through m. So, this is the transition condition. So, there are two important points here. The first point is that note that we have is element of sign over here, because this by definition what we have on the right hand side is a set of states, because in the case of a nondeterministic finite automaton given a state, and a symbol it produces a set.

So, we have a set on the right hand side. So, if r i belongs to this set then we say that the automaton goes from r i minus 1 to r i in the next step. And the second point is so r i is basically just some element, there may be other elements in this set, which are not r i which may be some other elements, basically we just means that it is just a different computation path.

And finally, we must write the acceptance condition. So, we accept if r m belongs to F. So, this is when we accept (Refer Time: 17:50). So, now with this definition, we are ready to define the language of a nondeterministic finite automaton. So, L of n is the set of all strings w in sigma star such that n accepts w. So, this is the definition of the language of a nondeterministic finite automaton. So, now, let us look at a couple of examples of nondeterministic automaton.

(Refer Slide Time: 18:38)


Eg 1 L = {w ∈ {0,1}* | 3rd last symbol of w is 1}
0,1
1
0,1
0,1
q0
q1
q2
q3
1110
1010
Defn A non determin finite automaton (NF is the five tuple
N = (Q, Σ, δ, q0, F

Consider the language L 1, which is set of all strings w over 0 1 such that the 3rd last symbol of w is a 1. So, I want to look at all those strings whose 3rd last symbol must be a 1. I do not care about the last two symbols. Now do I care about the first n minus 3 symbols? How can we design an automaton for this? Let us try to design a NFA, because we are studying about nondeterministic. So, what we will do is so we have our start state, let us call it q 0. So, what we do is from q 0, we try to do the following things, try to do the following non-deterministically. So, at q 0, I will always try to guess non-deterministically whether the next symbol is the 3rd last symbol or not.

There are two things that can happen either the next symbol is the third last symbol or it is not. Iif it if not the third last symbol then I will just ignore it. So, I will stay at q 0 on a 0 or 1. If it is the third last symbol then I will go to this state q 1 on a 1. So, if it is the third last symbol, I will go. Then again from q 1, I go to a state q 2 on seeing either 0 or 1, because if this is the third last symbol this will be the second last symbol I do not care what it is. And then from q 2, I go to a state q 3 again or a 0 or a 1, this is the last symbol and this is my accept state.

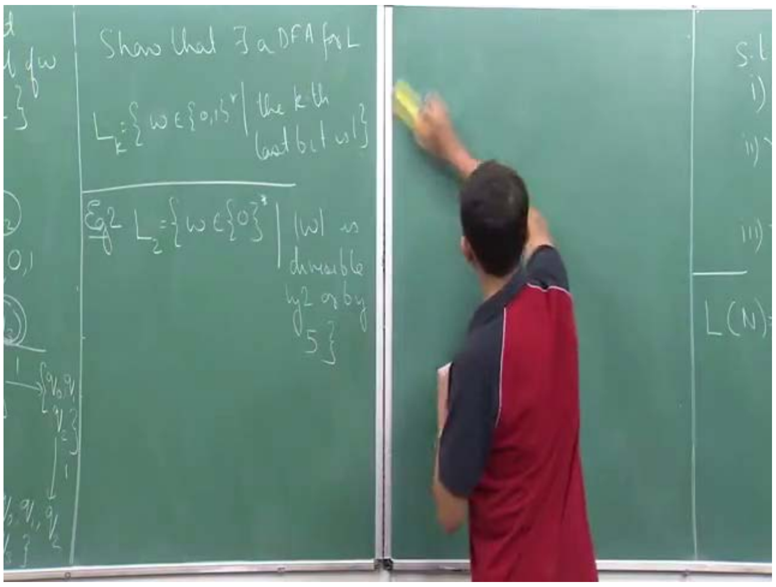
Now, let us look at some example. Let us say that I have a string 0 1 1 0. So, how does computation proceed for this string on this automaton? So, maybe I have change this, let us say I have a string 1 1 1 0. So, in this for this string, the third last symbol is this 1, which is a 1 which means that this should be accepted. So, what happens when we see this string? Let us try to create a table as we had done last time. So, when I see the first 1 I go to both these states q 0 and q 1. So, when I see 1, I go to from q 0 on seeing 1, we go to both q 0 and q 1; going to q 0, essentially means that I am guessing that the symbol that I am currently reading which is 1 is not the 3rd last symbol.

And going to q 1 means that I am guessing that it is the 3rd last symbol. So, I am using non determinism to do two things. So, when creating a clone of myself and one clone I am telling one clone that well the bit that I am reading is the 3rd last bit act accordingly; and to the other clone I am saying that it is not the 3rd last bit, it should also act accordingly.

Now, from q 0 q 1 on seeing the next one where do we go to, so again from q 0, we go to q 0 and q 1; and from q 1, we go to q 2. So, we go to q 0, q 1 and q 2. On seeing the third one, we go to again from q 0, we go to q 0 and q 1; from q 1, we go to q 2; and from q 2, and we go to q 3. So, although we have got an accept state; we cannot stop here, because the input is not completely read. Now the last bit is a 0. So, what happens when we read 0. So, on a 0, from q 0, we go to q 0 itself; from q 1, we will go to q 2. Let us write it down. So, I am seeing 0, from q 0, we stay at q 0; from q 1, we go to q 2; and from q 2, we go to q 3; and from q 3, we do not go anywhere. So, again that computation also dies out. So, this is the final set of states, and because q 3 is one of the accept states in the final set, hence I accept the input.

Now, if you take another example, so this is one example, if you take another example where the third last bit is not a 1, let say 1 0 1 0. And if you just carry out this computation, you will realise at the end, you should just try this out that you will not get q 3 in the final set of states, which means that this input will not get accepted. So, this is an example of an NFA for this language. So, the question is why do we construct an NFA for this language, or in other words can we construct a DFA for this language.

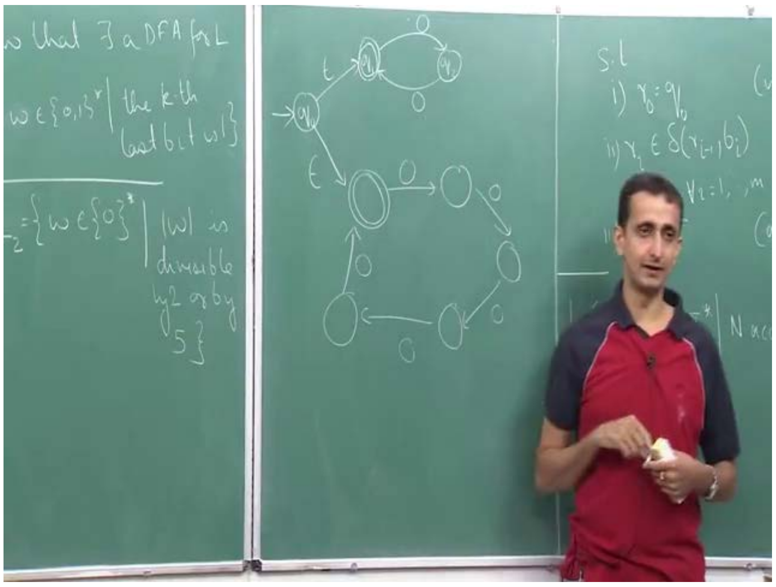
(Refer Slide Time: 25:26)


Show that ∃ a DFA for L
L_k = {w ∈ {0,1}* | the k-th last bit is 1}
Eg2 L_2 = {w ∈ {0}* | |w| is divisible by 2 or by 5}
L(N)

So, as a exercise, you can try to show that there exist a DFA for this language L as well. And you can try to analyse how many states do you need, what is the minimum number of states in the DFA you need that would accept L. And in general, what you can try is that if you consider a language L k, which is set of all strings over 0 1 such that the kth last bit is a 1. If you try to construct an N F A for it using the same logic, I mean you will realise that you can actually do this using just k plus 1 state using the same idea. But again just try doing a try constructing a DFA for this language, and then try to find out how what is the minimum number of states that you need. And then you can try to compare these two numbers, I means and see that what kind of advantage NFA gives.

Let us look at another example. So, this time, we will look at a unary language a language over just one alphabet. Let us look at the set of all strings over a single alphabet 0 such that the length of w is divisible by 2 or by 7, so how can you create an NFA for strings whose length is divisible either by 2 or instead of 7 let us take 5, 2 or by 5. So, here we will use epsilon transition to differentiate between these two ideas.

(Refer Slide Time: 28:14)



So, we start at a start state q 0; and from q 0, we non-deterministically decide what are we going to check. So, are we going to check, whether the length is divisible by 2 or are we going to check whether the length is divisible by 5. So, if we are going to check, whether the length is divisible by 2, what we need to do is we just have a automaton like this one is a 0, we go to this state. Let us call this as q 1 and q 2; and q 1 is our accept state. So, here if the number of 0s is even, I will be at q 1.

And the second case, I will basically have a 5 cycle with this being the accept state that checks whether the number of 0s is divisible by 5 or not. If a number of 0s is divisible by 5, then it would end at this state, otherwise not. So, I am using epsilon transition to decide what I am going to check, whether divisibility by 2 or divisibility by 5.

Thank you.