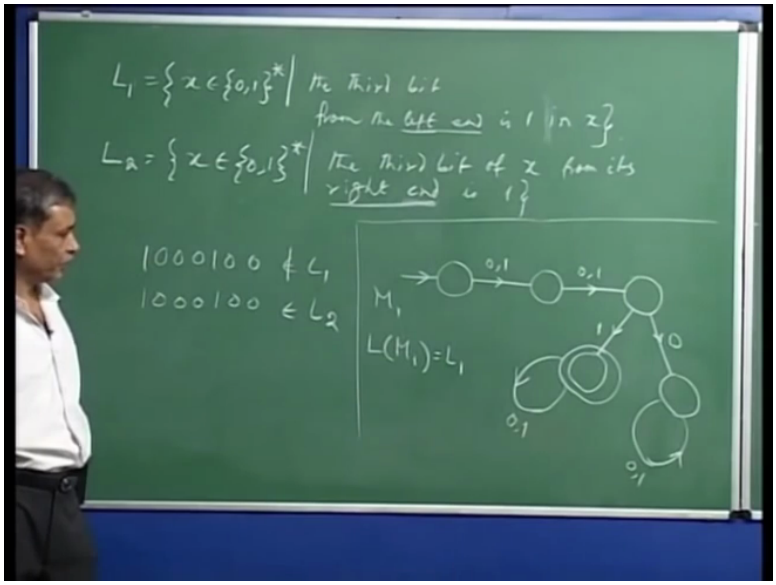
Theory of Computation
Prof. Somenath Biswas
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Lecture 5
DFAs solve set membership problems in linear time, pumping lemma.

(Refer Slide Time: 0:30)



Let us prove 2 more languages to be regular just to get our ideas about DFA and (()) (0:23) fixed in our mind, okay. These 2 languages are like this, the first language is again strings of binary over the binary alphabet, here we want that the third bit from the left end is 1 in x, okay and there is other language similar looking between see it is kind of any different in the implementation, you can say the third bit of x from its right end is 1.

So the 2 things are similar except here we want some condition from the left end in the first case L1 and here from the right end. So, so let us say I have this string 1 0 0 0 1 0 0, is this in L1? It is not because the third bit from the left end, so for L1 you are interested in the third bit from the left end this bit is 0 but it is not one, so therefore it is not in L1. On the other hand the same string 1 0 0 0 1 0 0 this if you look at from the right and the third bit is one.

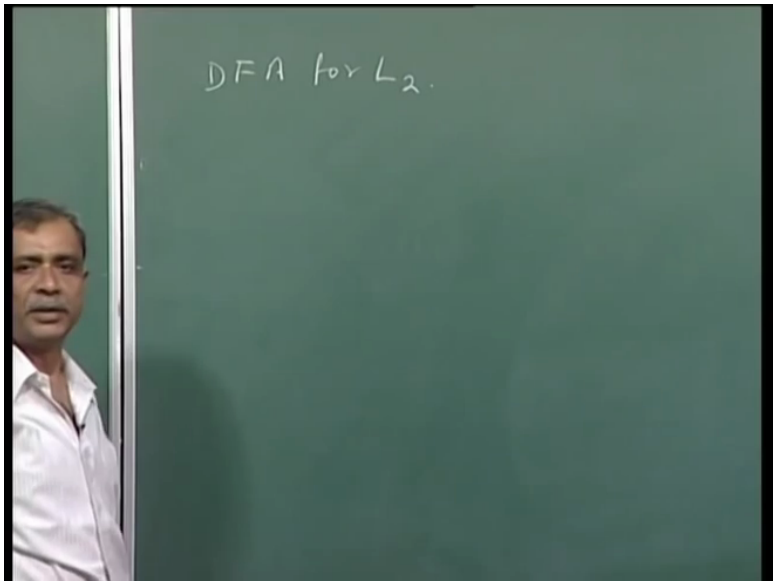
So strings satisfy the right end here, therefore this is actually in L2. How do I design DFA's for these 2 languages L1 and L2? Well L1 should be fairly simple, so what is the kind of information the finite state head should keep in its mind imagine you are looking at a binary string, starting from the left end. So the first bit is not of an interested you but you should remember that I have seen the first bit.

So when you see the next symbol you know it is the second bit again that is not of interest except to the point except to the extent that the next bit is going to be of interest to you. So you will agree with me that consider this machine M1 on either 0 or 1, it just goes to this state. So basically this state means I have seen no bits of the string when you come to this it when the DFA comes to this state it remembers that I have seen the first bit from the left then when the DFA comes to this state that means it has seen exactly 2 bits from the left end.

And now the bit that comes that would determine whether the input is in the language or not. So now if this bit is one, the third bit, right? 1, 2, 3, so now you should accept the string and what more? If that the third bit is 1 then it does not matter what bits come subsequently it should remain in this state which is the accepting state the final state.

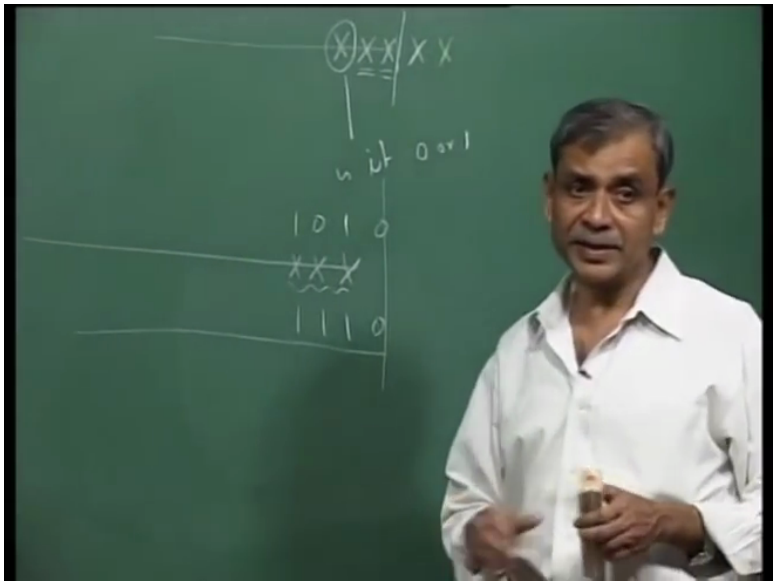
On the other hand if this bit was 0 then it goes to a state. Now any further augmentation on the bit screen cannot make the string in the language L1 because the third bit is 0. So it gets into a trap state and both for 0 and 1 it remains here and that is it. So I claim if this machine is M1 I claim that L of M1 is L1 and you will see that you will agree with me this is fairly simple to see we are just this DFA is looking at the third bit from the left and deciding whether to accept the string or not depending on the bit is 1 or 0.

(Refer Slide Time: 6:02)



Now let us consider the language L2, right? So you see the one problem we immediately face that we cannot go to the end of the string and then start back to check what the third bit from the right end was, right? So imagine this string, you cannot the DFA cannot do this, right? That it cannot go all the way to the end and then say oh! The input has ended, so let me trace back and check what the third bit from the right end was?

(Refer Slide Time: 8:11)



So what it should try to keep in its head? What of the current string, let us say the DFA has seen an input string up to this point, right? This is the last bit it has seen, it is the last bit it has seen, it is the last bit it has seen, what is of importance to the DFA? Is whether or not that this bit is it 0 or 1, right? If this bit is 1 then if the string ends here then that string should be accepted by the DFA that we are trying to design.

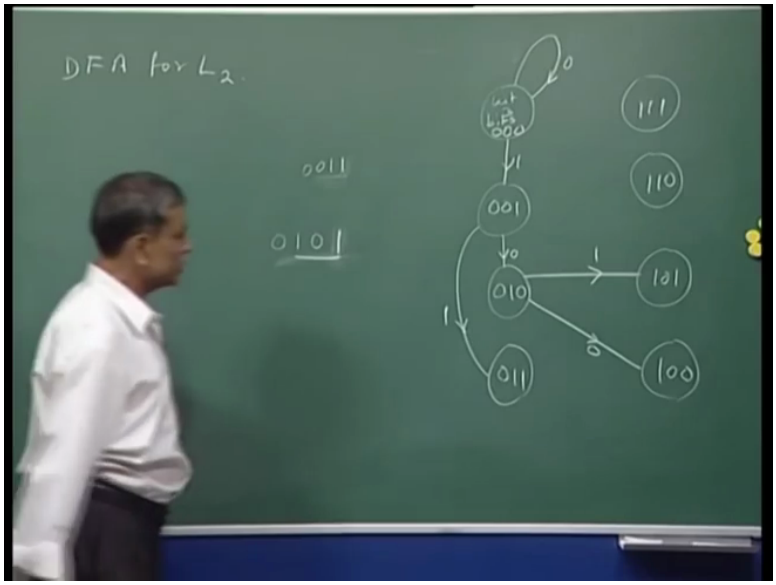
If that bit is 0 then that string is not going to be accepted. Problem is it is not just the third bit that you need to remember because you see, let us see after this, this is the part that you have seen of the input and this is not the end of the input, so another bit comes then which is the bit you should remember or which is of importance to you is now not this bit because now because of the addition of one more bit in the input string, the string that I have seen so far this is the bit which has become of importance and then if you see another bit, this is the bit which has become important.

So one can see, it is not difficult to see if you pause a moment that if I keep track of all the 3 bits that I have seen at the end of the string that I am currently looking at or in other words, supposing I have seen a string up to this point and what I should remember is the, these 3 bits, right? Now to remember 3 bits, how many different combinations of these bits can be there, that you know there are 8 possible combinations because this can be 0 or 1, this can be 0 or 1, this can be 0 or 1.

All these 8 possibilities must be kept track of separately because for example this was 1 0 1 and this bit was different, let us say 1 1 1 when the next bit came, let us say 0 came then if

this was the entire string and this the string ends here then this string should be accepted the one below, the one above should not be accepted because the third bit from the right is 0 here. So let us say we keep track of our DFA will remember what are the last 3 bits it has seen and in the sequence it has seen, right?

(Refer Slide Time: 10:04)



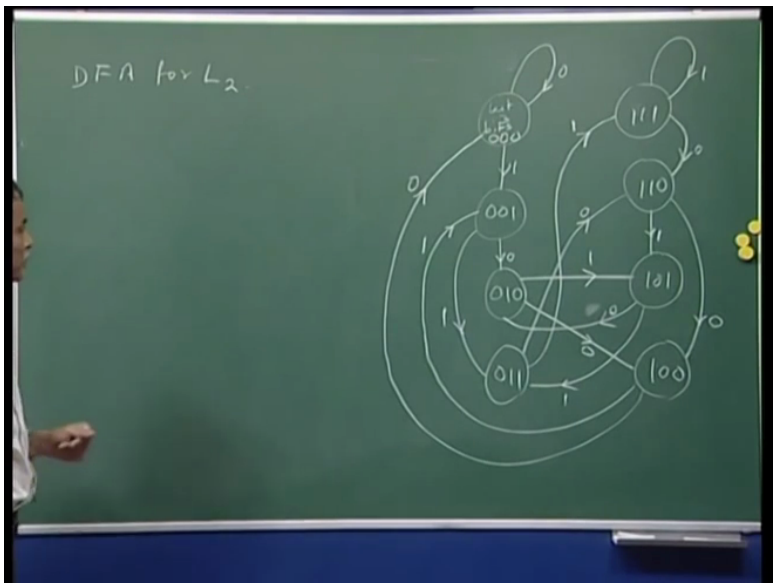
So in this case you can see it should, must have 8 different states at least. So let me write here, let us say this state means the DFA, what I am trying to ensure the DFA should be in this state, if the last 3 bits it has seen is 0 0 0, last 3 bits 0 0 0, here it is let us see 0 0 1, 0 1 0, 0 11, here it is let me put it this way 1 0 0, 101, 1 1 0, 1 1 1. So what is the goal? What is our goal for the DFA that we are trying to design?

That this DFA will remember the last 3 bits it has seen, the sequence of last 3 bits it has seen, if it has seen the last 3 bits that it has seen, if it was 0 1 0 it should be in this state and so on. So now let us think of the transitions, right? So now here 0 comes, so last 3 bits it is in this state if the last 3 bits were 0 0 0 and now 0 came. So which is the state it should go to on 0, clearly even now the last 3 bits that sequence is 0 0 0, so it should remain here.

On the other hand if your 1 came, last 3 bits it has seen will now be 0 0 1, so this comes to this state. Now we have taken care of this state and from this state, what are the? So 0 0 1 and now 0 came, so the last 3 we are talking about this particular state, if the machine was here at some point in the last 3 bits it has seen that sequence is 0 0 1, now 0 came so it is the sequence, now it has seen 0 1 0, so it should come here on 0 and on 1 it should go to 0 1 1, okay.

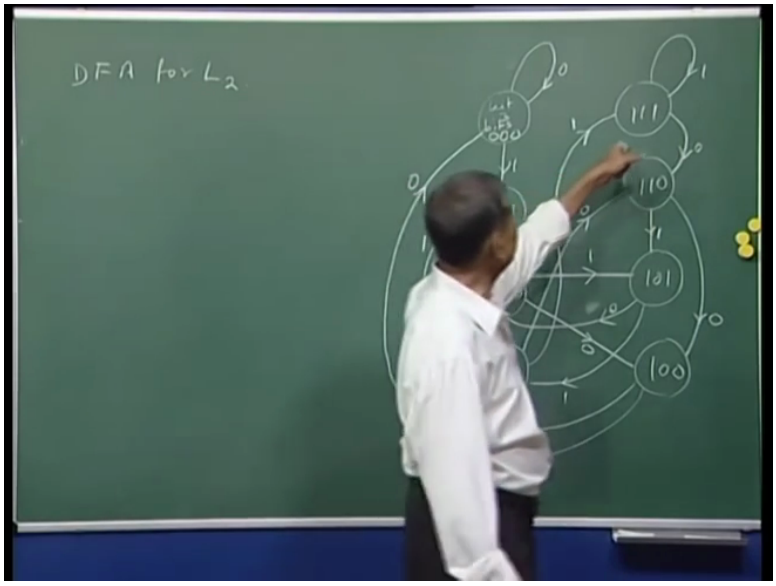
Then we have completely taken care of this state and now let us just consider one more state, so if this is the state the machine was in that means the last 3 bits it has seen 0 1 0, so now 0 came, so the last 3 bits became 1 0 0, so on 0 it should go to this state and if the last bit is, it is 1 it should go to the state 101 that remembers 101 to be the last 3 bits sequence and this way we can complete the diagram.

(Refer Slide Time: 14:04)



So this way when we complete the same things for all the states, our transitions from the state will like this, looks somewhat somewhat untidy but conceptually it is fairly simple that we have realized, now but the DFA is not complete I would need to say what are the final states? That is easy; a final state should be that state in which the third bit from the right and that it has seen so far is 1.

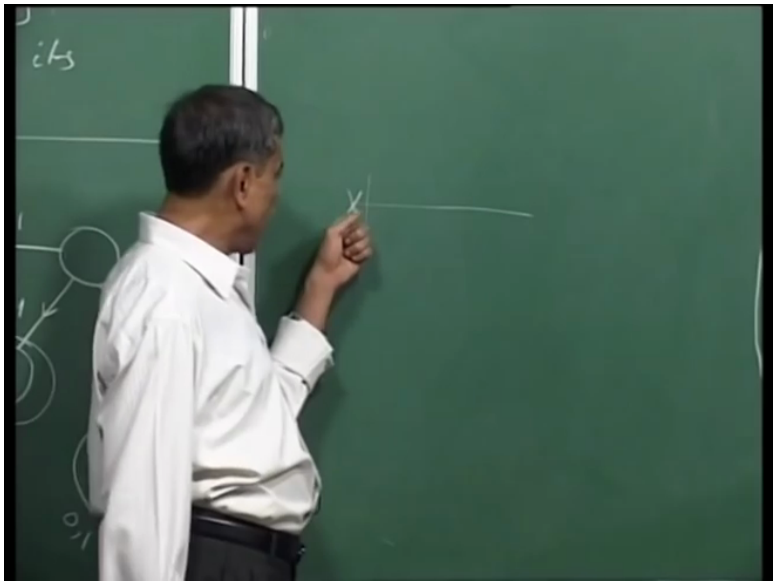
(Refer Slide Time: 14:36)



So for example if the machine was in this state then clearly the last 3 bits and the string ends there and string ends there and machine finally finds itself in this state then that string should be accepted because the third bit from the right end that would be 1. So you can see all those states in which the first of the sequence that we are remembering which is the third bit from the right end of the string that I have seen so far, with that is one it should be in accepting state.

So clearly then this should be accepting state and so should be this and this and this. On the other hand these 4 states are should not be accepting states, alright. Do we need more states that this? You might wonder that the meaning of these states that we have put here is this that the last 3 bits that I have seen but when I start, right? Supposing when the DFA starts it has not seen anything. So there is nothing like last 3 bits.

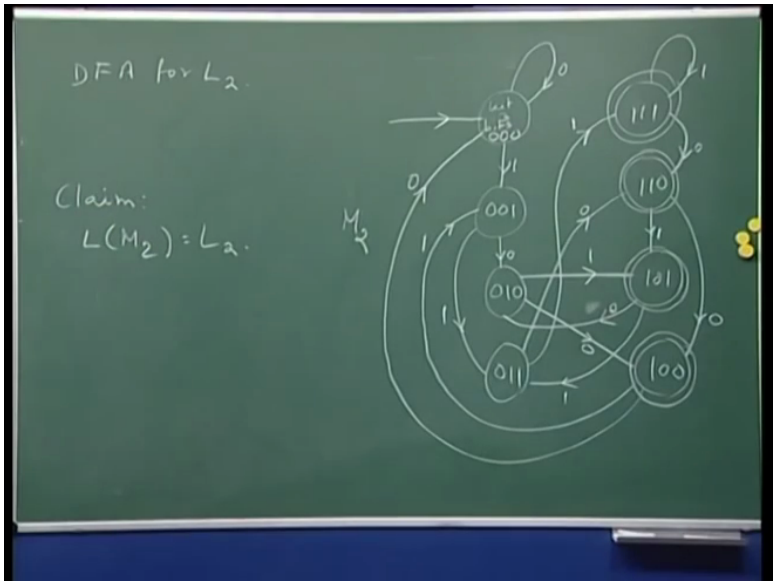
(Refer Slide Time: 15:56)



Similarly when it has seen just the first bit, it has seen so when it is here at this point having seen the first bit it has seen only one bit there are no 3 bits that it has seen. So what we do for this? You could have states for the initial part but as it happens, let me just say this that if you start here the thing is alright because although you have not seen the last 3, obviously when you are starting what you have seen is empty language empty string but it is alright to think as if you have seen 0 0 0, why?

Because whatever happens the next 2 bits, whatever be the next 2 bits, right? Next 2 bits will make, next bit will make these bits to be the last bit, another symbol comes then this will be the last bit, these on no 2 bits strings the machine should accept because the last, the third bit from the right is not 1. So you know, you can convince yourselves this is alright to keep this as the initial state.

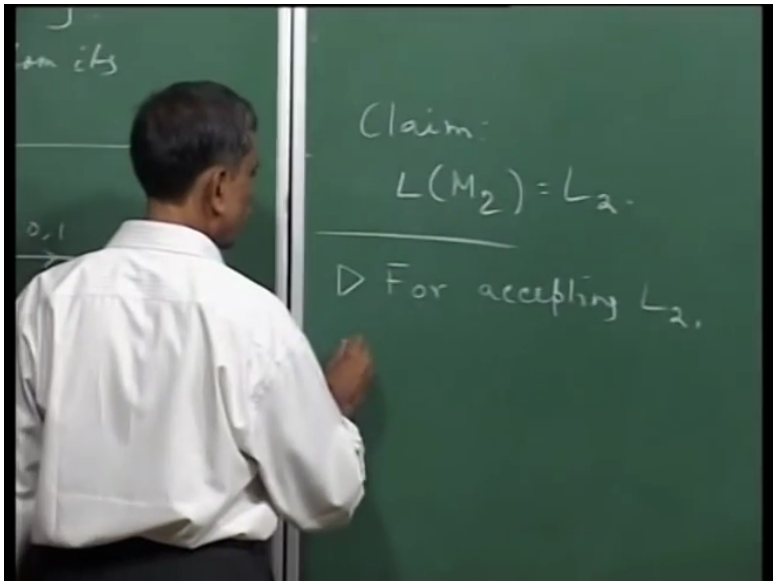
(Refer Slide Time: 17:53)



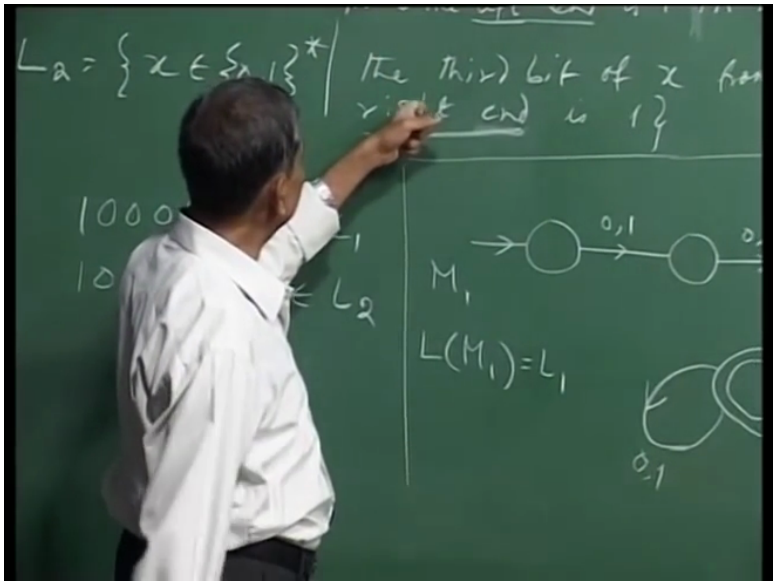
If I wanted, so that completes the description for a DFA for L2. So if I say that this machine is M2 then claim is language accepted by M2 is L2. Now again there is nothing very sacrosanct about the third bit, supposing I had said that tent with from 10^{th} right end similar machine I could have designed but if I use the same idea as was used here then how many states that machine will have which tries to accept all strings whose 10^{th} bit from the right end is 1, it will have to the power 10 states, 1024 rather too many.

Now you may wonder is it really necessary? Can we do for example could we have taken care of, could we have accepted this language L2 using a DFA with less than 8 states? The DFA that we design had 8 states, 2 to the power 3, 8 states because we are remembering all sequences of all 3 bits sequences which came at the end, that is the end so far encountered and therefore I have 8 states.

(Refer Slide Time: 19:50)

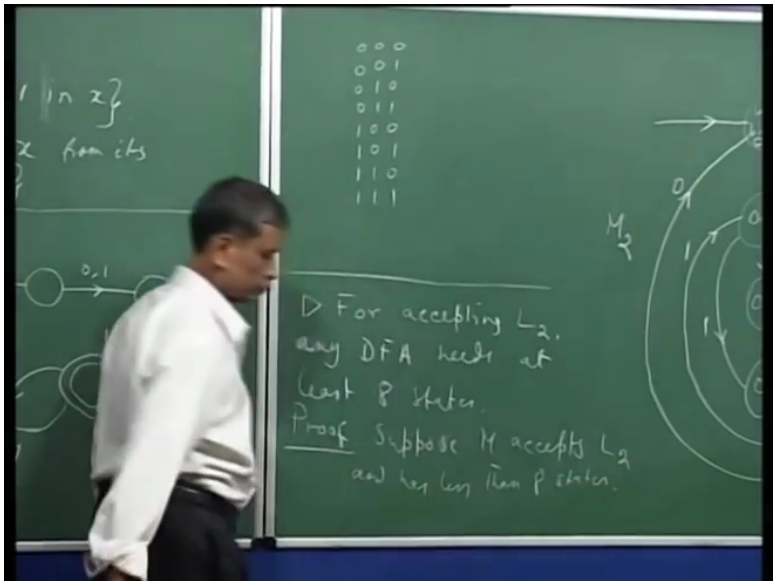


(Refer Slide Time: 20:13)



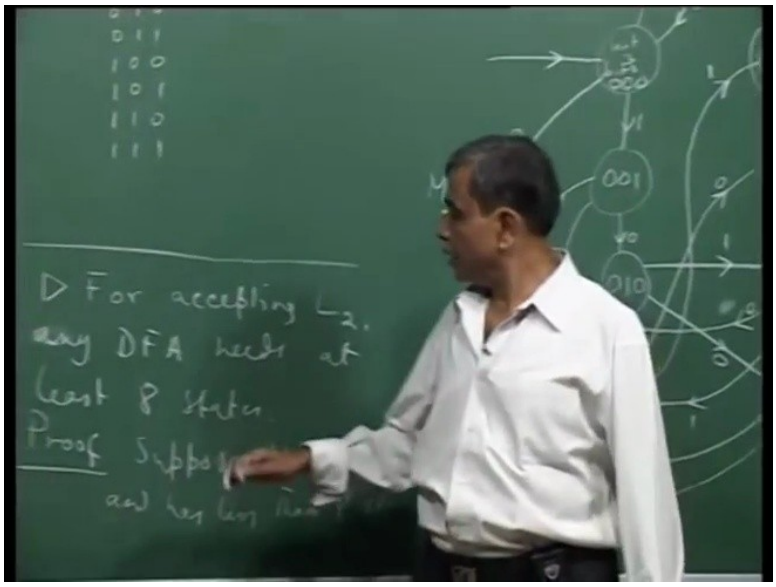
Could we have done with less number of states? And it is not too difficult to see that for accepting L2 any DFA needs at least 8 states and in fact if my language was instead of trying to check whether the third bit from the right end is 1, if it was 10th bit from the right and is one than I would have claimed that for accepting L2 any DFA would have needed 2 to the power 10 bridges 1024.

(Refer Slide Time: 20:33)



Let me just prove this statement for L2 that we need at least 8 states this manner. Just consider these 8 strings 0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 101, 1 1 0, 11 1 these are 8 strings, the 8 strings of link 3 binary strings and I am trying to prove this statement that any DFA that accepts L2 must have at least 8 states and we will prove it by contradiction, suppose M accepts L2 and has less than 8 states, okay.

(Refer Slide Time: 21:55)



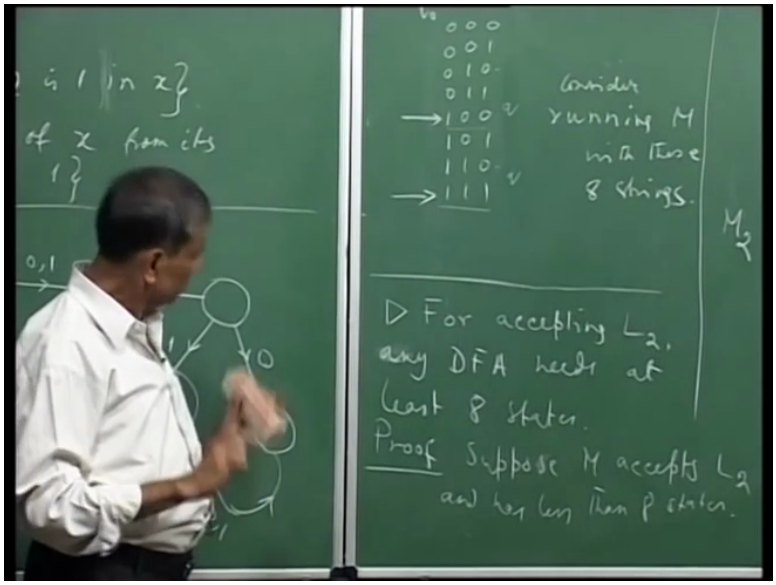
Now remember this machine M has strictly less than 8 states, so it has 7 or less states and here I have a list of 8 strings, so what I do is, this machine let us say M consider running M

with these 8 strings, so of course it will start with its initial state, M's initial state and let us say at the end it goes to some state q1, here it is q2 and so on, right?

So there are total number of states is 7 or less, number of strings we have 8, so now you remember from pigeonhole principle it means that there will exist strings you which will take the machine from the initial state to the same state, so then these 2 strings be like that? That this string 11 0 and 0 1 0 took M same state, so that is not possible because this string should be accepted that is string should not be and that state is either an accepting state, a final state or not a final state.

So it cannot behave differently that it will accept this and not accept this because the state is either final state or not a final state. So clearly these kinds of things are not possible but maybe you may say that let us say this string and this string took the machine to the same state. So let us say that is okay because both are final, both are strings with the third bit is from the right end is 1.

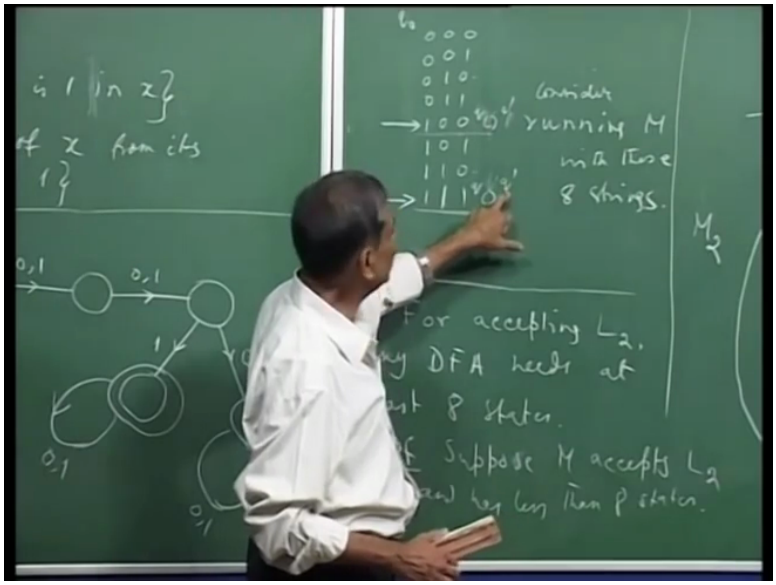
(Refer Slide Time: 24:41)



So both should be accepted, so that is fine, so suppose this string took the machine to q and this string also took the machine q, one thing you can assert that q should be a final state there is no contradiction so far but imagine now another bit comes, so let us say another 0 comes.

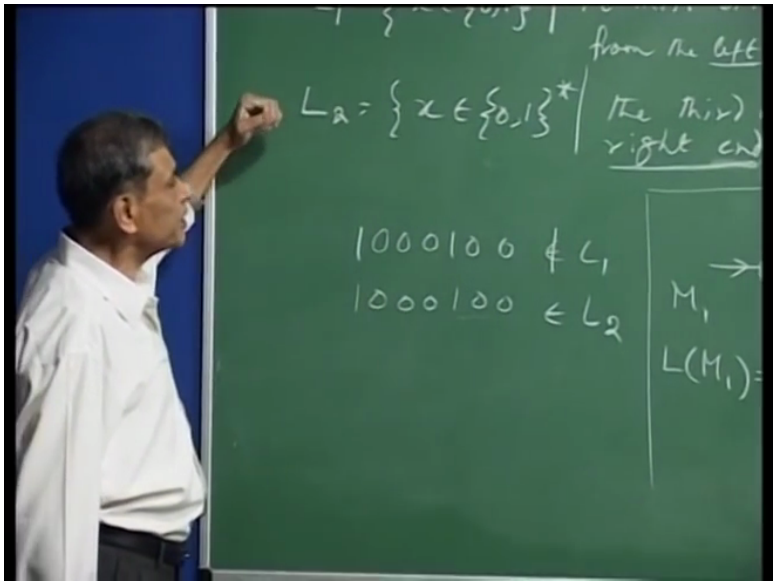
Now what is going to happen? Because here in both these cases the machine M went to the same state q and now 0 came, again it will go to the identical state let us say q dash, so q on 0 goes to q dash, some q dash and now we have a problem because now what should q dash be? Is it an accepting state or a non-accepting state? Because where is the problem this string should not be accepted.

(Refer Slide Time: 25:56)

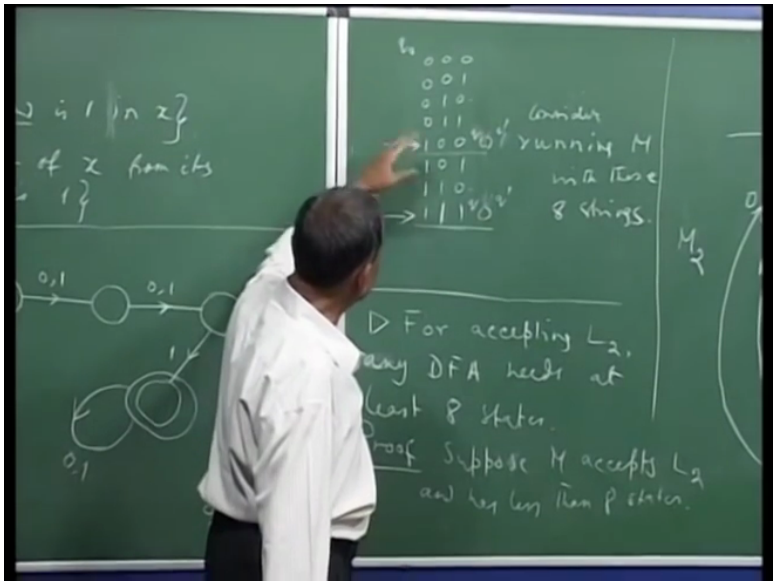


On the other hand this string should be accepted because this is we have one from which is the third bit from the right end and so we have a problem, we have contradiction for this I need q dash to be non-accepting non-final, here I need q2 be a final state that contradiction. So we got into the contradiction because we assumed that this machine M which accepts L2 has less than 8 states.

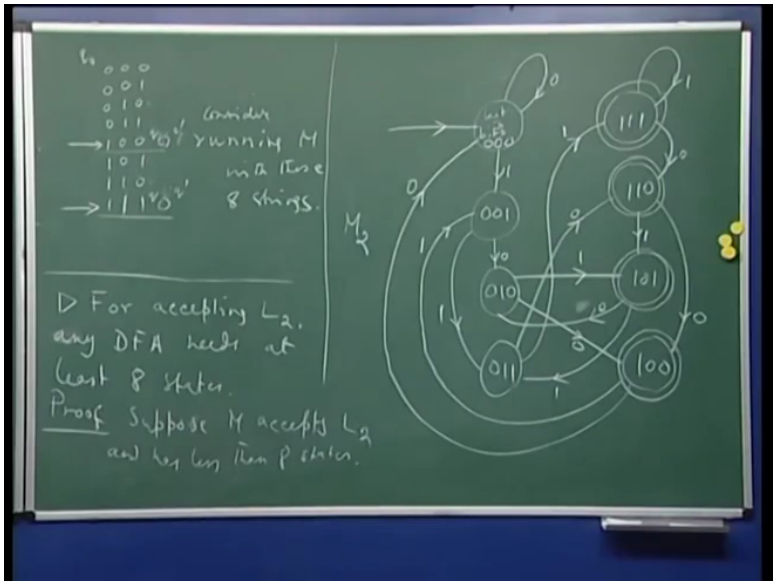
(Refer Slide Time: 26:19)



(Refer Slide Time: 26:34)

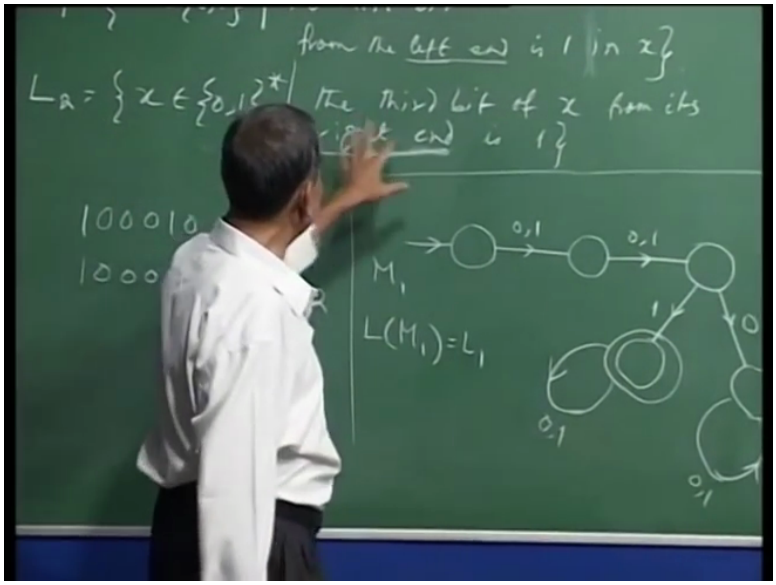


(Refer Slide Time: 26:53)



So therefore my conclusion is any DFA which accepts L2 must have at least 8 states, right? And therefore we can by the way you can you can this argument that I have given you can check that, that will be the case whatever be the 2 strings which take the machine to the, 2, 3 bit strings that take the machine to the same state, if the machine had less than 8 states it has to happen that 2 of these must take the machine to the same state and you can derive the contradiction.

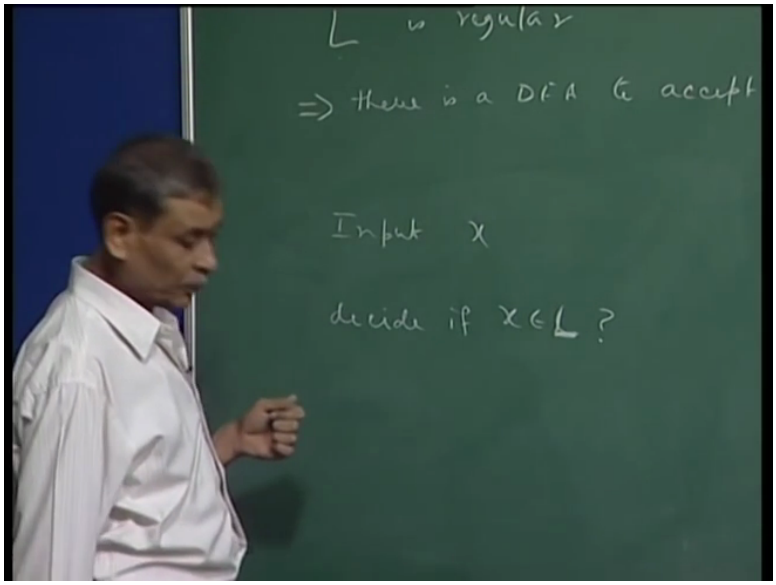
(Refer Slide Time: 26:58)



So if we had the language, another language where we are trying to see whether the tenth bit from the right end is 1 or not? Then by the same argument if we are doing it by DFA that DFA necessarily will have 1024 states at least. Now we have seen many examples of languages etc by DFA. We know that every finite language can be accepted by DFA that means every finite language is regular.

Then we have seen several examples of infinite languages which are also regular. We have also seen that if a language is regular then its compliment is also regular, if 2 languages are regular then their union as well as their intersection both are regular, so we have a fairly large class of languages, you feel intuitively which is that class of regular language is fairly large and at this point of course you will wonder, can languages for which I can decide the membership problem by means of a computer program, is it possible that all such languages are regular?

(Refer Slide Time: 28:44)

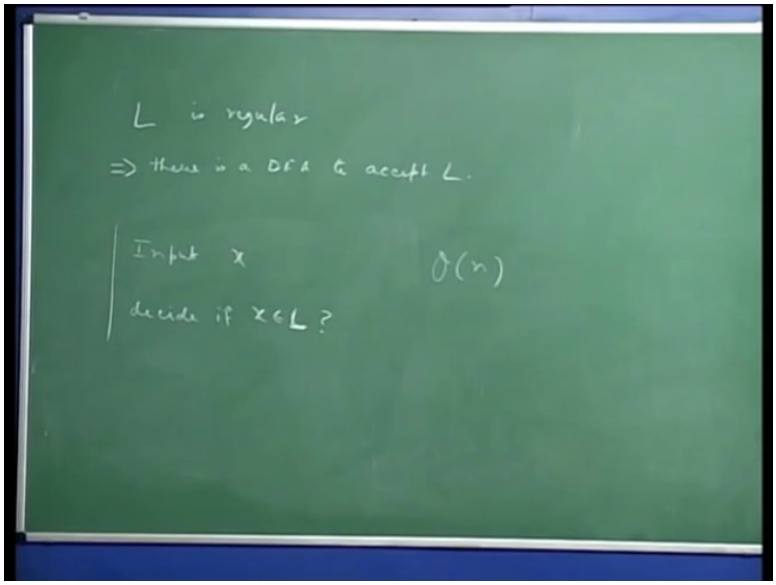


Now intuitively the answer should be no, why? Because you see suppose you say L is regular, in that case necessarily there is a DFA M to accept L. Now suppose I am interested in solving the membership problem of this language which is a set of course that means somebody will give me an input x which is a string and on both sides if x is in L this language. So this is set membership problem.

Now since L is regular, there is a DFA to accept L and given x I can just run the DFA on x. Of course you can very easily I am not going to those details but you can see you can easily write a computer program to simulate a DFA, that computer program will take symbol after symbol of the string from the left end and implement the DFA to check which is the state the machine would be having scanned some part of the string, for each prefix your computer program will figure out where the DFA will be, it is fairly easy to write that program.

And what will be the time complexity? And that program of course will solve this problem because when the when the program at the end of the input string reaches a state of the DFA which is a final state then it would no string x is in the language otherwise the string x is not in the language.

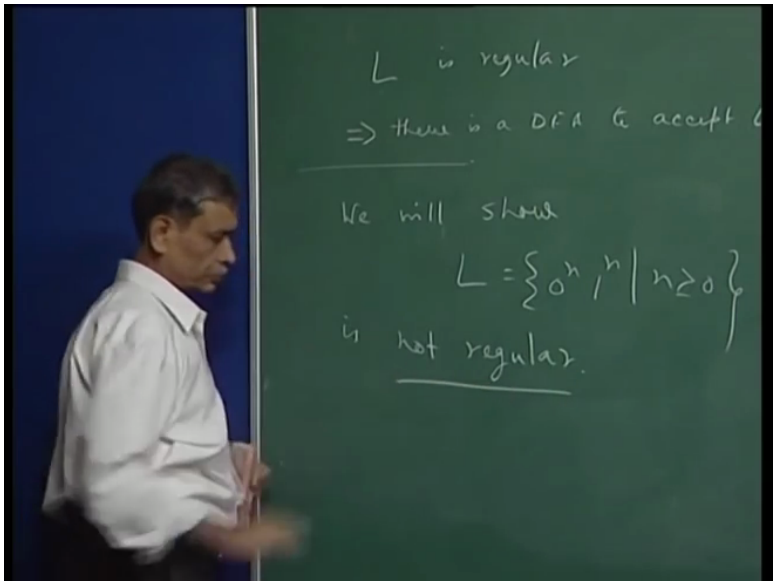
(Refer Slide Time: 31:31)



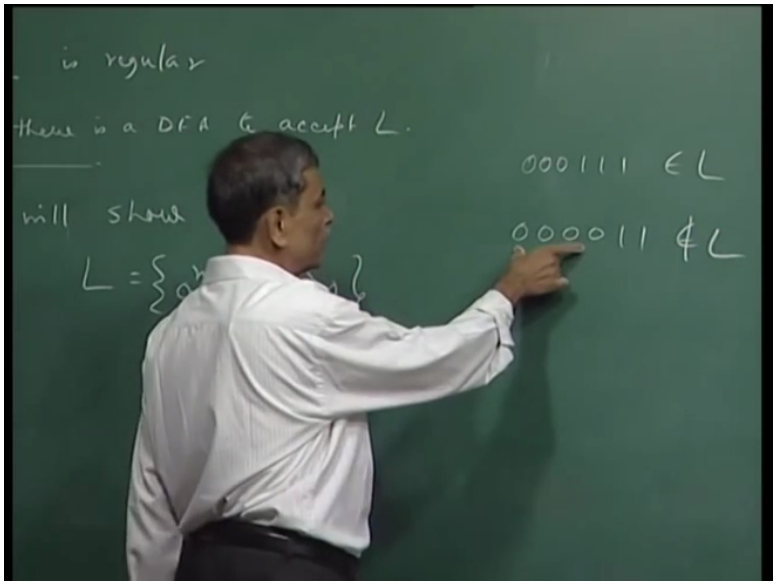
So this program and since you have studied some simple I mean some amount of algorithms that you know about time complexity, when you implement the DFA to solve this problem is that algorithm time complexity is going to be order in where n is the length of the input, here the input is the string, so that means now if you thought if somebody tells you that look I think that all membership problems can be solved by DFA's then you know you will realize that there is something wrong because all membership problems if they are going to take order anytime which is too much to expect.

You know you expect that some membership problems will be harder even if these membership problems are computable. Maybe it will take n (()) (31:52) time and square time and cube time and so on. So intuitively you will feel therefore that there will be many languages which are not regular, right? Even if we can solve the membership problem of such problems by computer programs and now we will argue, we will give you a methodology or proving concrete languages to be not regular.

(Refer Slide Time: 32:36)



(Refer Slide Time: 33:16)

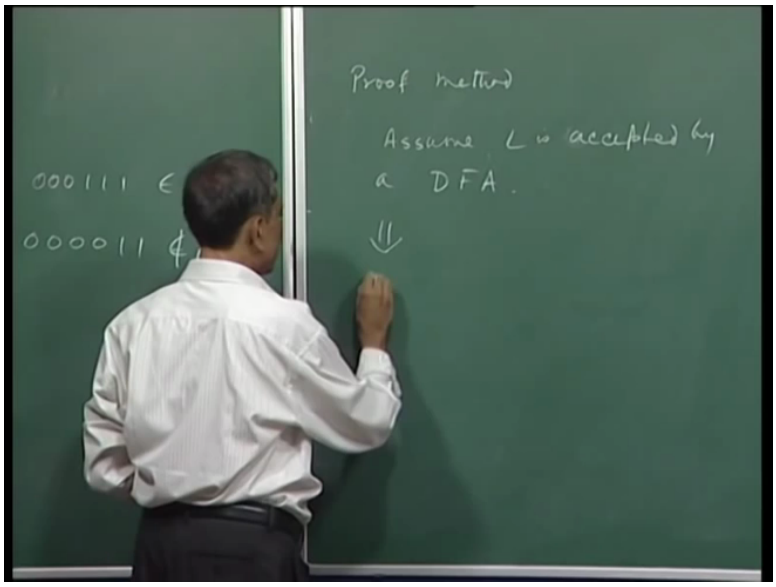


So for example you will prove, we will show that this language L which is 0n 1n. So basically these are strings where you have number of 0's followed by number of 1's only constraint is that the number of 0's is equal to the number of 1's. So 0 0 0 1 1 1 that should be in this language L. On the other hand 0 0 0 0 1 1 that this should not be in the language L because the number of 0's here is 4, here it is 2.

How do you show? In fact how do you show that a language to be not regular that means I should be able to prove that there is no DFA to accept let us say this language and one

standard method you can imagine will be that I will prove this statement, so we will show a complete this sentence, we will show that this L is not regular.

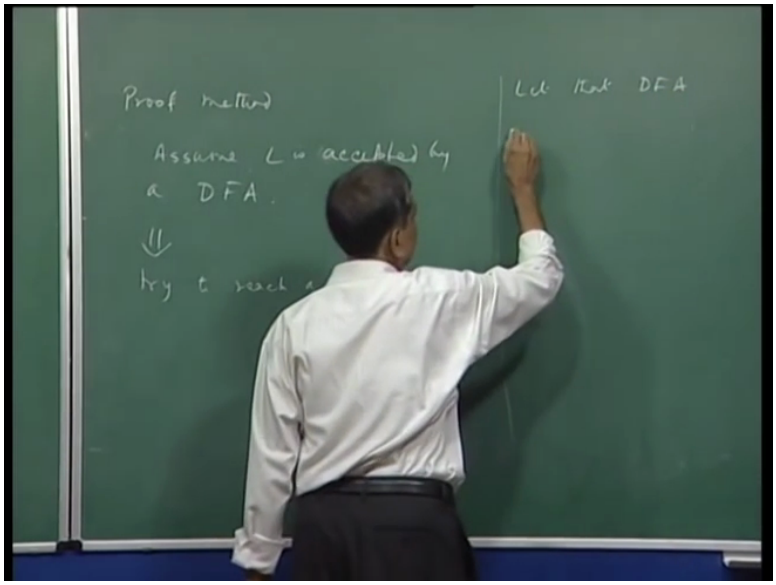
(Refer Slide Time: 34:05)



So proof method could be, likely to be that assume L is accepted by a DFA because if it is regular then we are giving a proof by contradiction and the we will assume that L is accepted by a DFA, we assume that L is regular, so therefore L is accepted by a DFA and then we should try to reach a contradiction. Well, we can intuitively see maybe this thing will work and indeed it works, this proof methodology will work out.

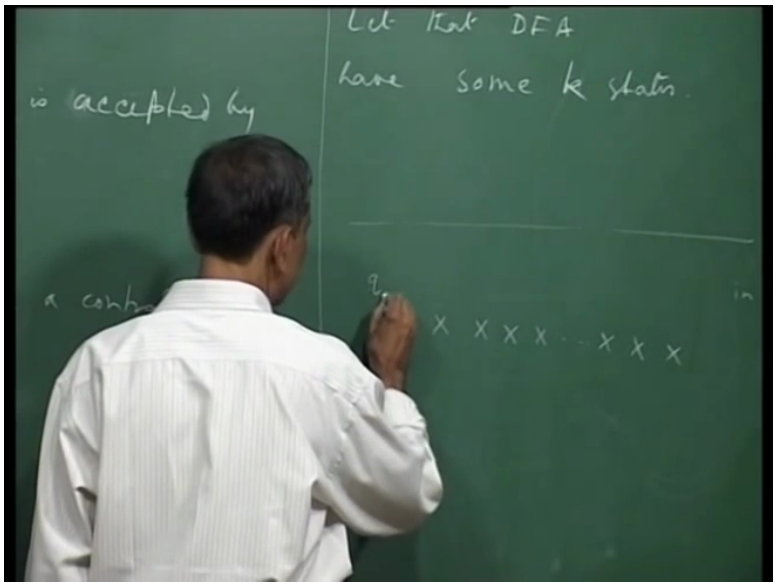
And let me give you intuitions behind such proves that see there is one severe limitation, once you say L is accepted by DFA comes about in this manner.

(Refer Slide Time: 35:30)



Suppose L is accepted by a DFA, let that DFA has some K states, right? Remember that DFA can have any number of states but once you say that this DFA accepts this language, so you have fixed a particular DFA and that DFA has number of states which is a constant. Now the DFA which accepts some language L it has K states and imagine there is a very long string which is in the language in L and consider the sequence of states the machine goes through.

(Refer Slide Time: 36:39)



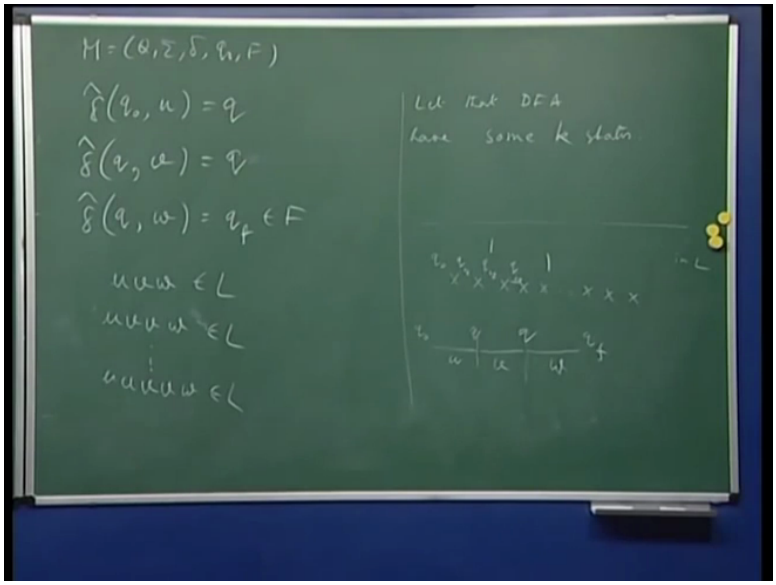
So remember that if you had supposing these are the symbols of the string, you will start with q0 and then let us say qi1, qi2, so seeing the first 2 symbols it is in this state and so on and here is another state and all that, so we have talked about it. Now if that string had let us say

to the length of the string is 2k and so this state sequence is going to be therefore length of the state sequence is going to be 2 K plus one, so then what that means?

Again by pigeonhole principle that here is the sequence which is much which is the length of that sequence is larger than the number of states and each element of that sequence of state is of course a state, what it means therefore that some 2 states must be identical because we just do not have so many different states to put states differently when the string is large.

So imagine a situation like this that I have a string and first part of the string, so this is a string I am thinking in 3 parts that this part took the machine from q0 to some q and since the string is long some states are going to repeat, so let us say this state is repeating that is first part of the string let me call it u took the machine from q0 to q, second pair part of the string v, let me call it v took the machine to the same state q because the state is repeating and then from here it went to a final state, so let me call this W, right?

(Refer Slide Time: 39:03)



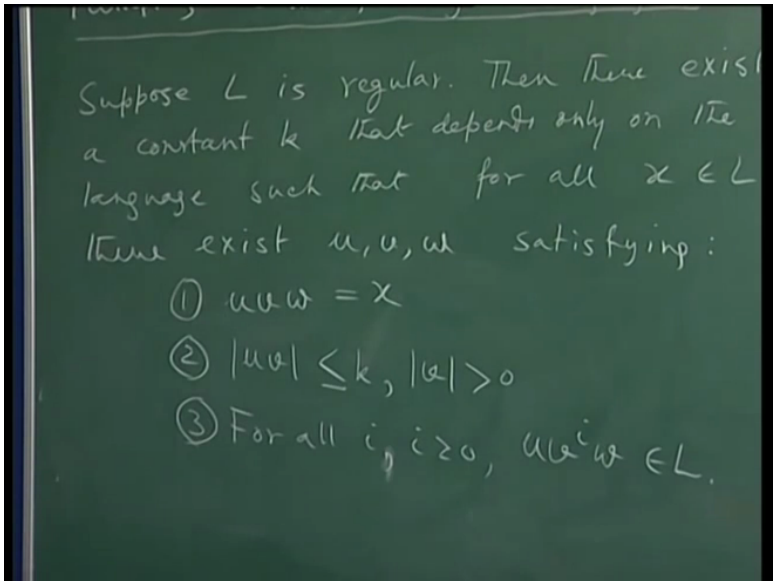
So the way I have drawn this picture, is it clear? So in fact let me use this that Delta hat of my machine M is let us say Q, Sigma, Delta, q0, F, so what we are saying through that picture is that Delta hat of q0, u is q, Delta hat of q, v is the same state q and Delta hat of q, w is qf some qf which is in a set of final states. Now clearly the string uvw takes the machine from q0 to one of the final states, so uvw is in the language L.

What about uvvw? Look at these 3 things u takes the machine from q0 to q, v takes the machine from q to q the first copy of v, second copy of v again it is a deterministic machine, it cannot remember that whether it is the first or second copy, so it has to behave identically on both the copies, so on this copy also takes the machine from q to q and now w comes, so this machine also will take this string will also take the machine from q0 to qL.

So therefore this string uvvw also will be in L and so on, uvv vw also will be in L and so on. So we can pump the string in which a state is repeating on which a state repeats and such strings can be pumped to get larger and larger and larger strings all of which will be in the language is the string is in the language. Now that is actually a severe limitation and we will see it, how that is used?

So this intuition that I just told u is captured by a lemma, what is called pumping lemma which is invariably useful for proving languages to be some non-regular? So let me state the lemma first and then before proving with the proof idea is like this, let me first I will give you an example usually we use a particular result known as pumping lemma to prove some language to be non-regular.

(Refer Slide Time: 42:08)



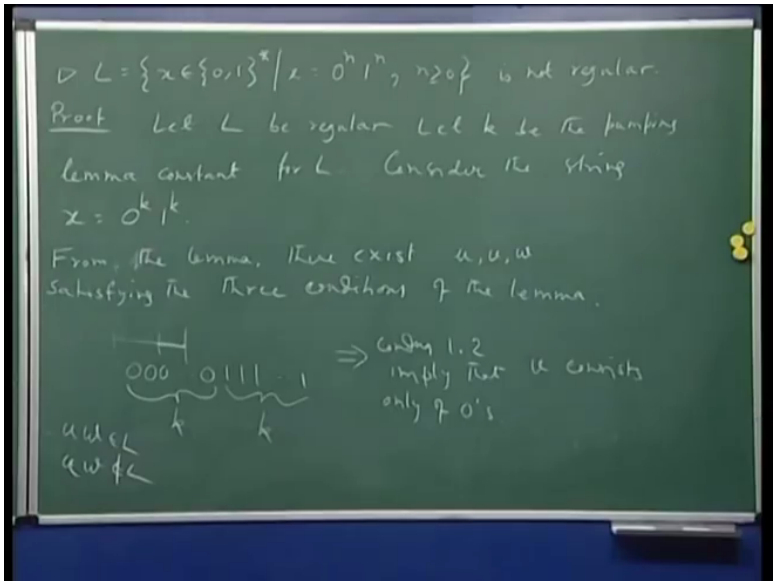
So let me state the lemma, pumping lemma for regular languages the statement is something like this. Suppose L is regular then there exists a constant k that depends only on the language L such that for all x, x is of course a string for all x in the language length of x being greater than equal to k we have there exists or let me state it like this, for all x in L, the length of x greater than equal to k there exist u, v, w which are again 3 strings satisfying condition one is that u, v, w that is the concatenation of the 3 strings u, v, w is actually the string x.

Second that length of u, v is less than equal to k and the length of v strictly greater than 0 that means v is non-empty string, third this is where the name pumping lemma is coming from for all i, i greater than equal to 0, u the string uviw is also in the language L, okay. Couple of things when I say that this constant k that depends only on the language, all we mean is that from language to language of course this constant may be different.

However once you fix a language this is constant that means it does not matter about the string that you are talking of and notice that remember our what we said right in the beginning that v to the power 0 also makes sense because i is greater than equal to 0, v to the power 0 is empty string that would mean basically you are saying that the string uw is in the language as well as uvw is in the language which of course begin with it was but now with i equal to 2 uvvw is in the language, uvvvw is in the language.

So another way of stating this is that you can pump this v any number of times, 0 or more times and yet the string that will get is guaranteed to be in the language L, right.

(Refer Slide Time: 46:48)



Before we prove this lemma let me show an example of its use L language we talked of little earlier, set of all binary strings such that x is of the Oh! This language is not regular. Let me prove this using the lemma, pumping lemma. The way these proves will go like this, Proof let L be regular, I will prove this essentially by contradiction assuming L is regular then I will invoke this pumping lemma to get a contradiction.

Let k be the pumping lemma constant for L, consider the string x which is k 0's and k 1's, now you can invoke the pumping lemma for the string because length of x is greater than equal to k, in fact the length of x here is 2k. So the pumping lemma says that there exist from pumping lemma from the lemma there exist u, v, w satisfying these 3 conditions, the 3 conditions of the lemma.

So this is your string, there are k 0's here and k 1's here and now what this lemma says that this string I can look at as a concatenation of 3 strings u, v and w. Now the entire u, v path that is the concatenation of uv is less than equal to k that means where, see this string I am saying is same as u, v, w do you therefore see because of this condition u, v to gather must be somewhere within zeros only because you have k zeros and length of u,v is less than equal to k.

So in particular condition 1's conditions 1 and 2 imply that v consists only of zeros because u,v itself consists only of zeros and therefore we necessarily will have only zeros and what more? v is a non-empty string, right? v is a non-empty string. Now consider the string on

pumping 0 times that means let us say we just removed v. So I will get u and w, according to the lemma this u and w is in the language, what did I do?

There was some v which is nonempty which I removed from this string of zeros, they were k zeros to begin with non-removal of v, length of v is at least one, so clearly in u,w can I claim this? It is clear in uw the number of 0's is strictly less than the number of 1's, right? 1's did not change because all we had removed was the string v and v consisted only of 0's, u also consisted only of 0's of course.

So 1's we did not touch, all we did was to take out a few 0's, at least one 0 from these k 0's and now I have a string uw which the lemma tells me that will also be in the language but now that string has number of 0's which is strictly less than the number of 1's, this string uw therefore cannot be by definition of L, L is the set of all languages in which 0's and 1's have the same number by definition of L uw, this string uw cannot be in the language, uw is not in the language.

Now your lemma says uw is in the language and definition of the language says uw is not in the language therefore you have a contradiction and the way to resolve the contradiction will be that to assert one of the assumptions that you have made and the proof that is false and only assumption you have made in all this, is that which could be false is L is regular, so therefore this proves that L is not regular, right?

So this is the use of this lemma. We have seen one example of how to make use of the lemma to prove some language to be non-regular? We will show some more examples of the use of the lemma and then we will of course need to prove that the lemma is with correct, so we need to give prove this lemma and there are some simple variations of the lemma we should also talk about that. We will do that in next lecture.