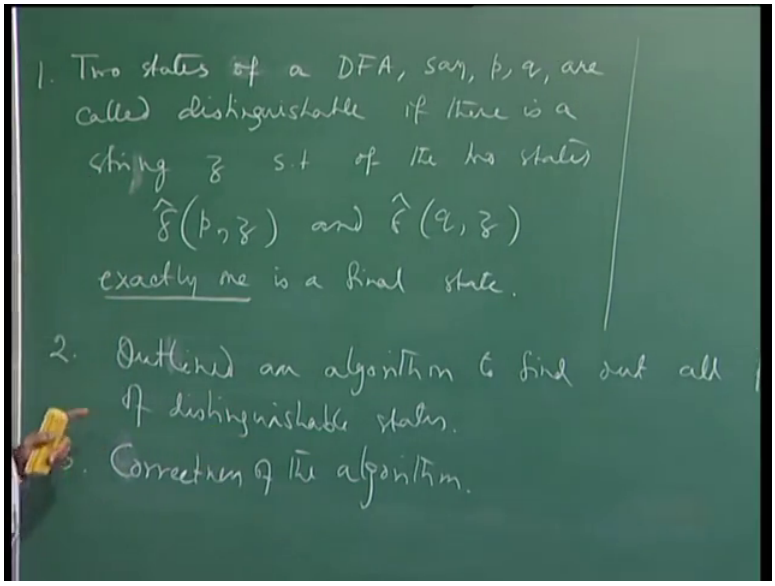
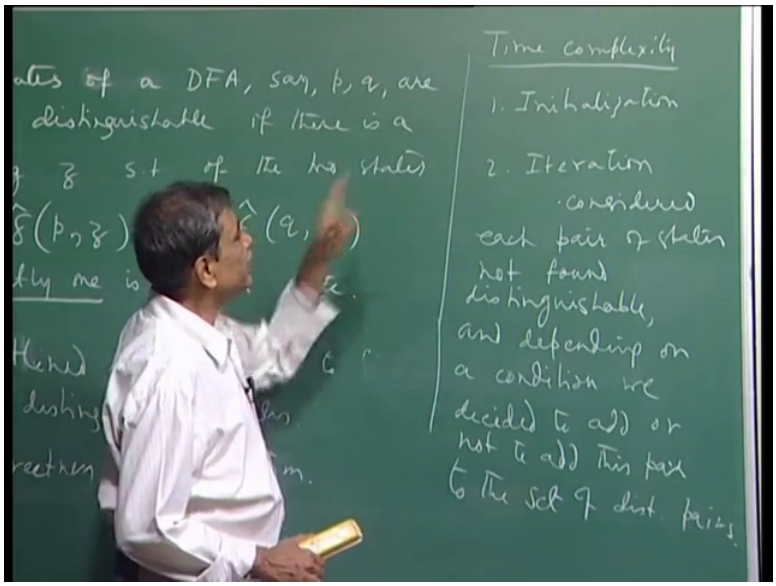
Theory of Computation.
Professor Somenath Biswas.
Department of Computer Science & Engineering.
Indian Institute of Technology, Kanpur.
Lecture-19.
DFA minimisation continued.

(Refer Slide Time: 0:37)





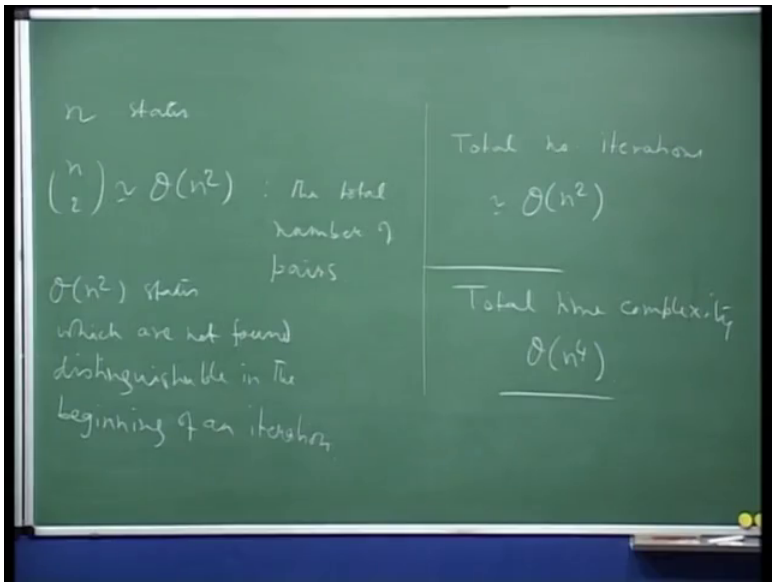
Let us recall what we did last time very quickly. 1st of all we defined the notion of when 2 states of a machine M to be called distinguishable? Let us write that definition. 2 states of DFA say p and q, p and q are the 2 states are called distinguishable if there is a string, string z let us say such that of the 2 states, Delta hat p, z and Delta hat q, z, so of these 2 states exactly one is a final state. In other words if z takes the machine from state p to a final state, then the

same z must take the machine from q to a non-final state. And if that happens for all z, if there is, if there is, part and if there is one string z, which shows this kind of behaviour with respect to p and q, then we say the pair pq is distinguishable.

So that is one notion that we did. Then this is of course the definition, the other thing we did was that we outlined an algorithm, outlined an algorithm to find out all pairs of distinguishable states and then we proved that the algorithm is correct. Today let us begin by considering the time complexity of the algorithm that we had outlined earlier. So we recall the algorithm what we did, we are talking of time complexity. 1st of all there was an initialisation phase in which any pair in which one state is a final state and the other is not a final state, such place we made them to be, we called them to be distinguishable and they are of course distinguishable because the z there is just Epsilon.

So that was the my initial set of states that we definitely know to be distinguishable. And then there was not iteration process and in the iteration process, in each iteration we considered every pair of states which, which is not yet found to be distinguishable and then we, so that 1st was initialisation and 2nd was iteration and in iteration what we considered was, considered each pair of states not found distinguishable, right. And depending on a condition, we decided to add or not to add this pair to the set of distinguishable pairs. Without going into the details we can see this, the initial phase, what did we do, we took every pair in which there is one state is a final state, the other state is not final state.

(Refer Slide Time: 7:27)

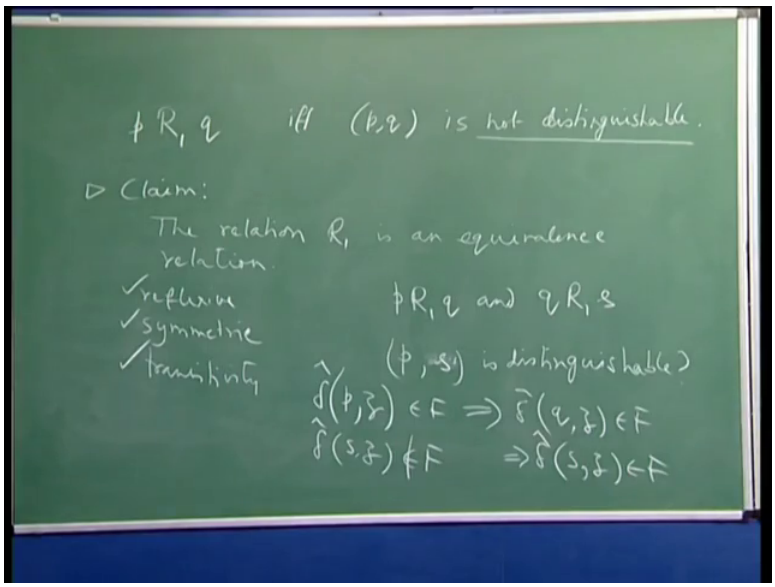


And how many such pairs can be there if there are total N states, total number of pairs, unordered pairs that are possible of these N states is of course N choose to which is order N square, this is the total number of pairs and as you can see in the initial process, initialisation in the worst case it will be about order N square and each iteration, what we are doing, we are considering one already not found distinguishable states, everyone of these states and then checking a condition. So in general there can be ordered N square states which are not found distinguishable in the beginning of an iteration. And for each pair what we do is a constant amount of work.

If you go to the details of what we do, the condition, you will easily see that for each pair of such states which are not yet found to be distinguishable we do something which is a constant amount of work. So, and how many such iterations will be there? Clearly total number of iterations is bounded by again order N square, why? Because in each iteration we definitely add one or more pairs to the set of distinguishable pairs set. So therefore since there are totally so many pairs to begin with and in each pair we are adding at least one, so at most there can be so many iterations. In each iteration we are doing order N square one so therefore total time complexity is order N raised to 4.

So order N 4 is time complexity of the algorithm that we have outlined. I should also mention that if you do a little bit more, then this algorithm can be improved to order N square. Let me just indicate what is that extra thing that you might like to do, so that time complexity improvement happens. And that is this, that suppose in an iteration I take a pair which is of course not found distinguishable till then, remember, then what we do is see we have such a pair p, q and then we consider one simple after another and say for example on I a p goes to p1, q goes to q1, the machine goes to q1 from state q on a.

(Refer Slide Time: 12:50)



Now if this is not already in the set we just discard it. But you see suppose in some clever way we can keep this information that if ever this pair p1, q1 is found to be distinguishable, then pq also should be marked as distinguishable, is not it. Because whatever string that distinguishes p1, q1, if you had a in front of that string, that would distinguish p and q. So keeping this idea one can design an algorithm which will improve the time complexity to order N square. Now, at the end of the algorithm you of course have discovered all pairs of states which are distinguishable. Consider a relation R on again set of states, let me call it R1 and we say that 2 states p R1 q, this, they are related by this relation if and only if the pair p, q is not distinguishable.

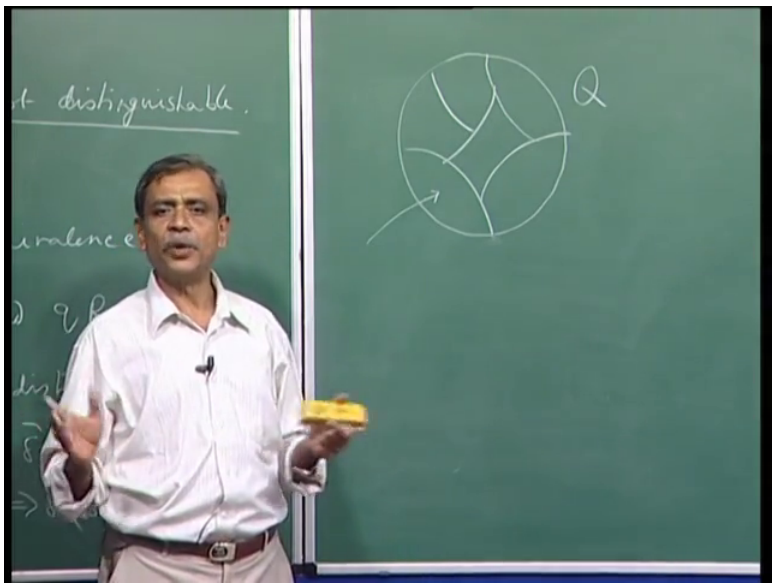
The 1st point is that this relation R is an equivalence relation. The relation, this relation R1 is an equivalence relation. And this is fairly easy to check, reflexivity is of course true that p and p, that pair is of course not distinguishable. There cannot be any string Clearly which takes p, the 1st p to final state and the next p to a non-final state, it is after all the same state, both the states are the same state. So clearly it is reflexive and symmetry is also easy to see, right. On, what about transitivity? So let us say that p and q are not distinguishable, so they are related by this relation and q R1 s, p is not distinguishable from q, means pair p, q is not distinguishable and q, s is also not distinguishable.

Can it happen that p and s, this pair p, s is distinguishable? If so, of course in that case the relation R1 is not transitive. But suppose p and s are distinguishable, then that means what, let us say that there is some z, such that Delta hat of p, z is a final state and Delta hat of s, that is not a final state. Right. This is one possibility, p, s is not distinguishable, is distinguishable

means one such case will happen. But you can easily see that if p is a state which takes the machine or from which the string z takes the machine to a final state, then it must be the case that Delta hat of q, z will also take the machine to a final state, in fact to the same final state or to at least to one of the final states.

Why? Because, suppose this is not true that p, z takes the machine from, from p to a final state and that same z takes q, takes the machine from state q to a nonfinal state. In that case, the pair p, q would have been distinguishable. So therefore I can say that Delta hat q, z must be in final state, one of the final states and now use the fact that q is not distinguishable from s. So this actually also would mean Delta hat s, the same z, because of the same argument takes the machine to a final state but that contradicts this assumption. So therefore the relation R1 also must be an equivalence relation, I mean it is transitive and therefore the relation R1 must be an equivalence relation.

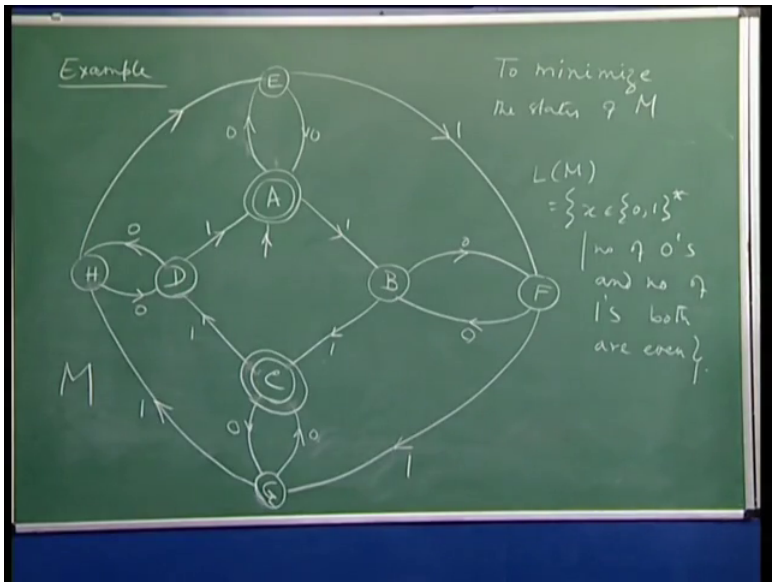
(Refer Slide Time: 17:53)



So now what we do is, given the set of states, we use this relation R1 to partition the set of states. Remember an equivalence relation will partition the set of states into a number of equivalence classes, right. And what we will do is that for each equivalence class of the set of states will have one state in our machine, the machine, the best machine that we are trying to decide and justification for this should be apparent from our discussion on Myhill Nerode theorem, right. So essentially what we are doing, using our algorithm, 1st of all be found out all pairs which are distinguishable, from that set we also came, it is easy to see which all pairs of states are not distinguishable, that when used as a relation is an equivalence relation, that equivalence relation partitions the set of states in equivalence classes.

And for every equivalence class of such states will have one state in the final machine that we are going to build, with a proviso which I will come to later, but at this point it is good to take up an example and see what is happening for the entire process.

(Refer Slide Time: 19:54)



Consider this machine, this DFA which we call M and we would like to minimise the states of this machine M. We come to another DFA with minimum number of states which would accept the same language. Now this looks complicated but we can argue about what is the language accepted by this. 1st of all notice that the initial state is this A and then on 1 you go to B from A, another one you will go to C, then D and then back to A. And on 0, from any such states will go to another state and then come back here, then here from B you are going to F on 0 and from F on 0 you will come back and so on. So and you will accept if at the end of the string the machine is either in A or C.

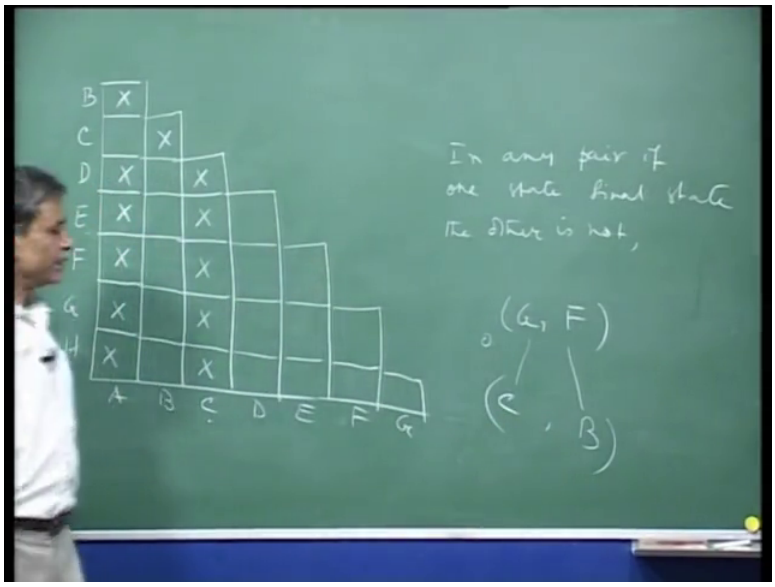
Now what is this machine trying to remember? I can see that this machine is trying to remember what is the number of 1 modulo 4, right. Like for example if the total number of 1s the machine has seen in an input so far, if it is in this state, you should be able to see that that number, total number of 1s should be, when you divide by for the remainder would be 3. And actually the same thing here also because you come to these kind of states, these outer states only on 0 is from one of these states. So here the number of 1s seen is 0 modulo 4, it is 1 modulo 4, 2 modulo 4 and so on. And here the same thing, 1 seen here again is 0 modulo 4.

But what about 0, in 0s you just have to remember that what is the total number of zeros is seen, whether it is even or odd. And it is, if you, if we consider it a little carefully, it is quite

clear because we are accepting here and here that means language accepted by M will be the set of all strings 0, 1 star such that number of zeros and number of ones, both are even. But of course for this language we had given right in the beginning, one of our 1st or 2nd class, we had given a very simple 4 state automaton, 4 state DFA. And we also proved that time by an ad hoc argument that there can be no DFA accepting this language which will have less than 4 states.

So in that automaton that we had provided, that automaton already had minimum number of states. Now that we have outlined the Myhill Nerode theorem, etc., can we use whatever we have learnt to go from this DFA do that earlier DFA 4 states that we had outlined long back, provided long back. Now 1st of all remember of a 1st step in doing this minimisation is figuring out which all pairs of states in this are distinguishable. Right.

(Refer Slide Time: 24:38)



For that one way of writing that will be, you know suppose I have, there are 8 states A, B, C, D, E, F, G, H, so let me make a table like this. This is just a way of keeping track of all pairs without bothering about their order A, B, C, D, E, F, G and you see what I mean is, we will make a table like this and what we will do is we will put a mark. For example if we find by our algorithm that the pair AG or GA equivalently, that square is distinguishable, then we are going to put a cross here. Let me just complete this table. Right. So what is our 1st step, our initialisation for finding out all pairs of distinguishable states?

It is to say, any pair in which one state is a final state and the other is not. So in any pair if one state is final state and the other is not, then that pair will be right in the initialisation

phase will be considered distinguishable. Now here which are the final states A and C, so as A and C, right. So A therefore is distinguishable from H, from G, from F, from E, from D, and of course from B. And similarly B is distinguishable from C because B is a non-final state and C is a non-final state. Anything else that we can do? So for example let me write it here, so C and this, D and here C is distinguishable from, so C is distinguishable from H, C is distinguishable from G, C is this equation will from F, C is distinguishable from E, C distinguishable from D, right.

And anything else that we can put? So the C column I have completed and A column I have completed and C whatever the column here I have completed. So these are the only process that I can put to begin within the initialisation phase. Now we will start the iteration, this was the initialisation phase and the iteration. Now take a pair like let us say G and F, right. In the initial phase, the initialisation phase, the initialisation, we will not, we did not find this to be distinguishable, so that is why we did not put the mark.

But consider what happens is that when we provide a 0 to this. So just consider what happens, G on 0 goes to C, so let me write it here, G on 0 goes to C. And F on 0, F on 0 comes to B, now you remember that this pair C, B was already found distinguishable right in the initialisation phase. Why? In fact that is what your of course C was the final state, C was a final state and B was not. So we already knew that there was a cross here, C and B, so from here when I considered this pair G, F in the next iteration, on 0 I found this is the pair that results and there is already a mark there indicating that this pair is distinguishable. Then I will make in that iteration G and F to be, this pair to be distinguishable.

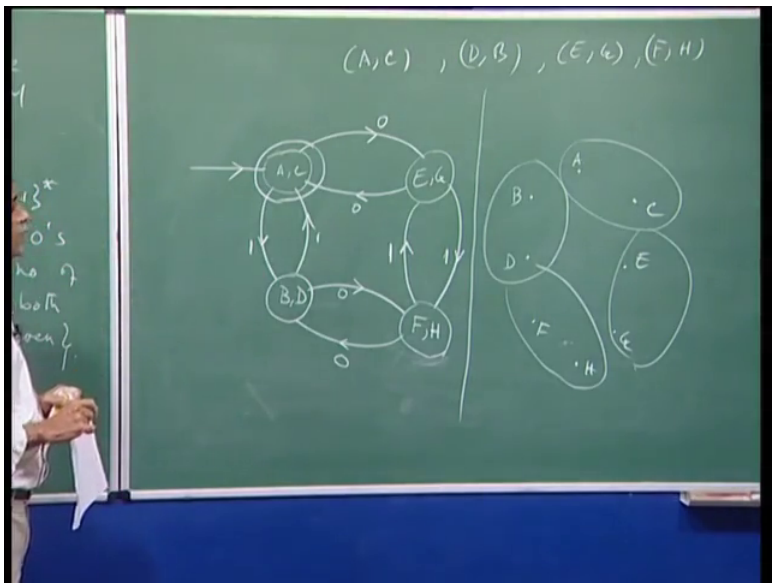
So now I will put another cross here, right. And if way we will keep on going, for example let see in the next iteration, F will go to B. Now remember that this pair C, D was already found distinguishable writing the initialisation phase. Why? In fact that is what you know the C was the final state, C was the final state and B was not. So we already knew there was a cross here C and B, so from here when I consider this pair G, F in the next iteration, on 0 I found this is the pair that results and there is already a mark there indicating that despair is distinguishable. Then I will make in that iteration G and F to be, this pair to be distinguishable. So now I will put another cross here. Right.

And this way we will keep on going, for example let see the next iteration, will, when you consider the pair F and E and then consider what happens on the input symbol 1. You see from F on 1, you go to G, so this is and from E on 1, E on 1 you go to F, right. Now the point

I am trying to make is that having discovered that G, F is distinguishable, we have put a mark, in the next iteration when we considered F and E, this pair F and E, when I consider the symbol 1, this is the pair, already there is a mark, so F and E will get a cross here and so on. I did not give all the crosses that would have come in the 1st iteration.

Now when we complete all the, you know all the iterations and when we go find an iteration in which where we have not added any extra pair into our set of distinguishable pairs. That means in N iterations when I could not put one more cross, then the algorithm stops. At that point this table will look like. Now if you look at this completed result of our algorithm, then what do we find that A and, there is no cross in this position, so we know that A and C, they are not distinguishable because had they been distinguishable our algorithm would have put MR there. Similarly I know D and B, they are not distinguishable and here it shows E and G is not distinguishable and F and H is also not distinguishable.

(Refer Slide Time: 35:27)



So now if you go back to our relation y1, what did we say, that we take the set of states of this machine in this is the set of states will have A , B, C, D, sorry, let me just write it this way, A, B, D, C, then E, G, F and H. So what we found was that these 2 will be the same equivalence class and in fact there will not be any other state in the same equivalence class, so A and C, B and D, F and H and G and B, these are the 4 equivalence classes, they have partitioned my set of states. And now what we are going to do is to consider an automaton where that automaton will have one state for each of these equivalence classes.

We are outlining the process of building the minimised automaton with an example from this automaton. Now what we did that we had one state for every equivalence class, so A and C. Now what will be the initial state of this new automaton that I am building? Whenever, whichever equivalence class in which the initial state of the original machine was there. So A was the initial state in this original machine. So therefore this is the initial state of the machine that a building. And now we have to fill up the transition. So let us say from this state, which is the state the machine would go to on 0? All right, what happens on A on 0, it goes to E, right.

So therefore it is sufficient, you just consider what happens on one of these states and so on E on 0, this machine went to C and we check the equivalence class of the state in which, the state E was there, the equivalence class, so this is a state which corresponds to E, G, E is here, so 0, we are here. And so similarly from A on 1 where do you go, A on 1 you go to B, so we will put an, put this transition here. And now if you see, from E, where would you, from this state on 0 where would you go to? So let us just see from E on 0 where do we go to, we go to A, A is here, so in fact will come here. And E from this state, where do we go to on symbol 1?

Well, from E you go to symbol 1 you go to F. You know it should be obvious that instead if I had taken G, here I just looked at on E, where does the machine go on 1 and we said E 1, the machine goes to F and so therefore this is the equivalence class in which let F is there, therefore we put an arrow here, the transition 1. But what would have happened if we had taken G? From G on 1 where do you go to, G on 1, we go to H, so G on 1 we would have gone to H and actually we would have put the same transition and that is not an accident or something, if you think about it, that is what should happen, right.

And similarly from, so this we have expanded the state completely and now from this state on 0, where should we go to, so F, from F on 0 we, in this machine we come to B. So F on 0 will come here and this machine is an F on 1 you will go to or H on 1, 4th will be same, H on 1, right, you go to E, so of course E will come here. And you should be able to complete it, it will look like this and which should be, which are the states here which should be called final states? Wherever A and whichever states in here, equivalence classes where either A or C would find themselves. But as it happens here, both A and C, then in the same state here, then this is the only final state that the machine has.

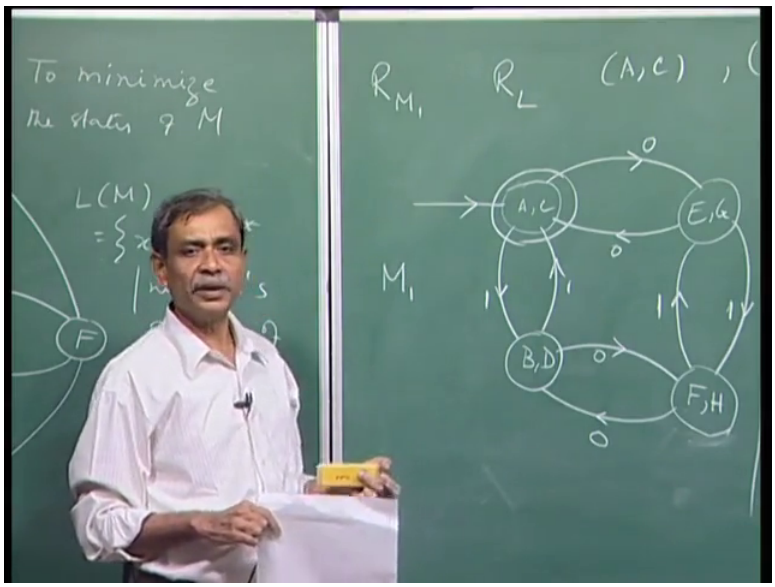
Now do you see that this machine is of course exactly the same as the machine that we had provided earlier for the same language and then of course we had proved by ad hoc means

that this is the best machine that we can do for this language. And the purpose of the example is that we carried out the algorithm and came to an automaton. In general what may happen is that after doing this process you might find some states in this, if I may call this reduced machine, in reduced machine which are of course, with the reduced machine I will have states, the equivalence classes of like this, of the set of states of the original machine.

So if there is any state such that that state is not reachable from the initial state, then from the so called, this machine that we are building, we remove all those things which are not reachable from the initial state and the resulting automaton that we will have will be the machine which is the best machine that you can have for accepting the language that we have. So to complete, let me just summarise their algorithms for minimisation. 1st of all you start with the process of finding out all pairs of states in the given automaton which you would like to minimise, the states you would like to minimise. So you use our algorithm to find out all pairs of states which are distinguishable.

So once you find that, then you make a partition of the set of states of the original machine and that partition will, each equivalence class in the partition will consist of all the states which are not distinguishable. Then you will have a machine in which every state of the machine corresponds to one of these equivalence classes, as A, C here corresponded to this equivalence class, this state corresponded to this particular equivalence class. And then provide the transitions in the manner that we described and then finally remove any, remove any state which is not reachable from the initial state. Identify of course, you have identified already the initial state to do this, remember initial state is that state, the corresponding equivalence class contains the initial state of the original machine.

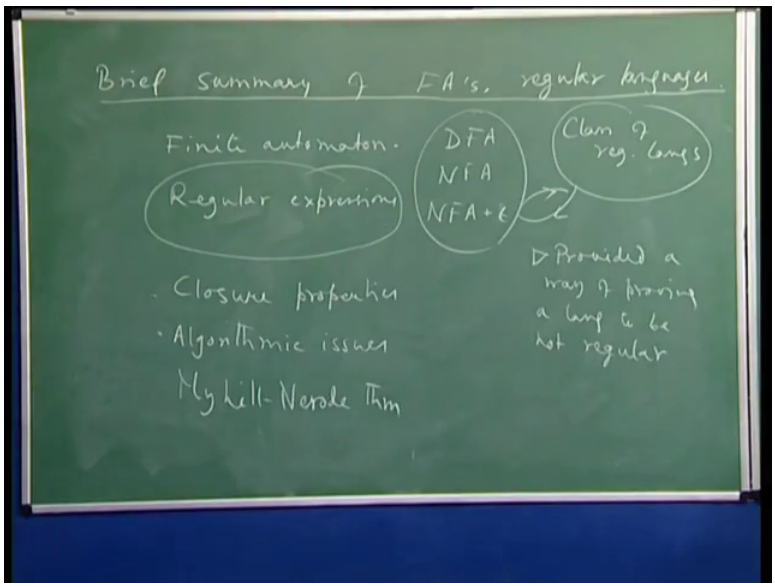
(Refer Slide Time: 45:22)



And you also found out what are the final states, these are all those states in this machine which corresponded to equivalence classes which had members of the final states of the original machine. And having removed the inaccessible or not reachable states, the resultant automaton will be the automaton with minimum number of states accepting the language that you, that this machine, the original machine accepted. And what you can prove, it is not too difficult that for this machine M, if I call this machine as M1, the relation RM 1, remember there is an equivalence relation one can define using on of course the set of strings of the language of the alphabet, this RM 1, this is actually the same equivalence relation as RL.

That means these 2 are, they partition sigma star identically. And therefore this machine will be the machine which has minimum number of states, which is something we proved from our Myhill Nerode theorem when we proved that the number of partition, number of equivalence classes of RL has to be less than or equal to the number of equivalence classes given by any such, at any machine accepting the same language. And that actually proves that any machine whose RM1 partitions sigma star in the same manner as RL does, that machine has to be the minimum state machine, the unique minimum state machine for accepting the same language.

(Refer Slide Time: 47:05)



We have completed all that we wanted to say about regular languages in finite automaton, so this is the time for a very quick recapitulation of what we did in all these many lectures. 1st of all we defined the notion of finite automaton and our 1st introduction in automaton was the deterministic version of finite automaton which we call DFA and then we considered some variance NFA, nondeterministic version, NFA with epsilon transition, let me call it NFA + Epsilon. At the same time, right in the beginning we had defined regular, the class of regular languages, by definition a language is regular if there is a DFA to accept it. Then what we found was, of course it was easy to see that NFA is a generalisation of DFA and NFA with epsilon transitions is a generalisation of NFA, what we found that all these machines, they accept precisely the same class of language.

So all these corresponded to the same class of language. So given a regular language, either you can make a DFA or NSA or and NFA with epsilon transition, similarly given from here we can go back to a regular language. Then we also studied regular expressions which is another way of expressing or denoting a regular language. Every regular expression we showed denotes a regular language and we also showed that for every regular language there will be a regular expression. So this is yet another way of capturing the same class of languages which are regular languages. After that we studied closure properties and we found for all kinds of operations like union, intersection, right, complementation and then we considered things like reversal.

For all these many many of the operations, the language class is closed. That means if I take 2 languages, apply that operation to get another language, that new language will also belong to

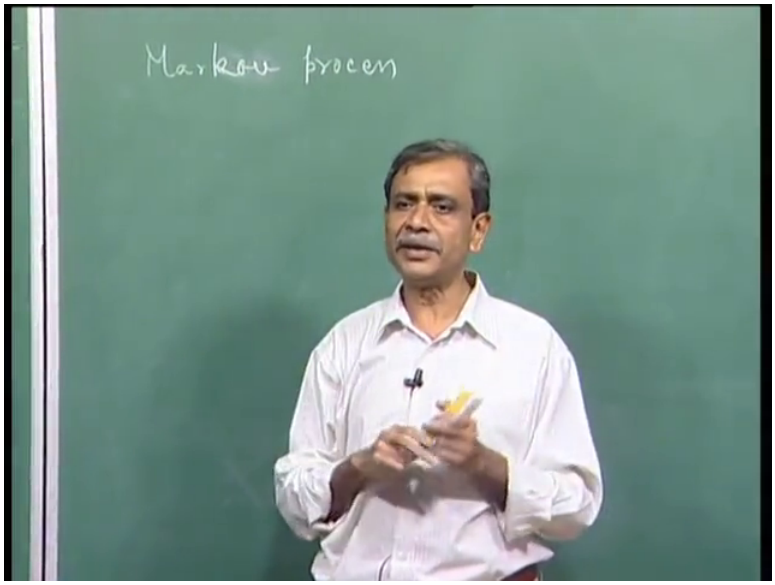
the same class. Then we had also look at some algorithmic issues like what will be the algorithm for getting, starting from a regular expression and going to a DFA for example which accept the language denoted by the regular equation and so on. Also certain decision problems if we consider, that whether or not you know he language provided by DFA or NFA or one of these is actually the empty language

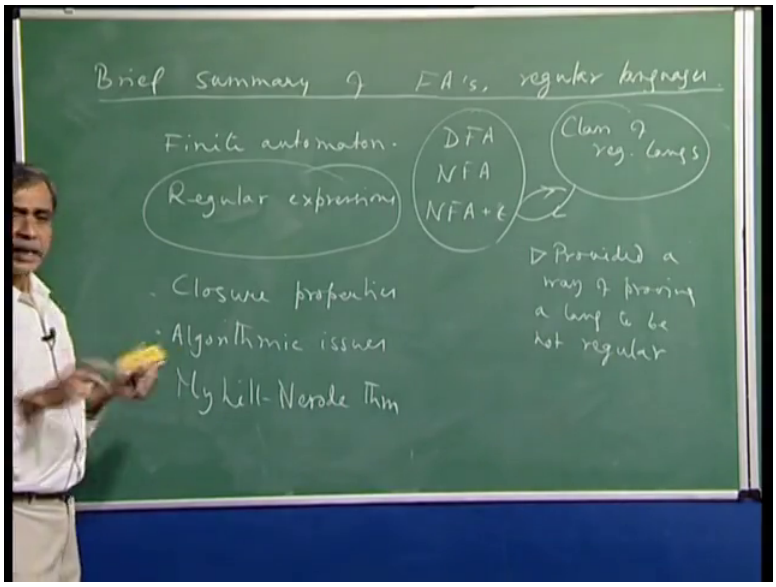
Now somewhere right in the beginning we did another very important thing which was, we provided a way of proving a language to be not regular. So you see the point is, if I want to show some language to be regular, it suffices for me to provide a regular expression to denote that language or provide one of these automaton to accept that language. But how do I prove that something is not regular? We gave a method, we used you know pumping lemma to show that you know certain languages are not regular, so that is one method of doing it. And then finally in the last few lectures we looked at Myhill Nerode theorem and showed how this Myhill Nerode theorem gives us say of minimising states of DFA.

I should mention that no such efficient algorithm is known to minimise the states of an NFA. Of course you might say why do not you go from NFA to a DFA but then that process itself in general can be exponential type. So we do not know of any efficient method of minimising states of NFA. In fact that ties up to a big issue in computer science theory, we will talk about it now. One final point I would like to say that this, you know this whatever we have learnt, finite automaton, regular languages, these are extremely useful in all kinds of branches of computer science. So the point is that this automaton later on you will see that either this, these kinds of things or they are generalisation certain generalisation that people do, they are very useful in many many branches of computer science.

In particular you will see that finite automaton, they are used for lexical analysis phase in compiler. And I should also mention that there are many other things which in which we capture the notion that some process is there and there only finitely many states.

(Refer Slide Time: 54:26)





For example I will not go into details here, Markov process, this is, this tries to capture probabilistic processes which have finite number of states and there is not much memory. You see the whole point of machine like finite automaton that we have seen, that the machine is one of these state and that is the only thing that it remembers about its past that I have reached the state or that state, no more. So a probabilistic process now which is basically uses that idea is called Markov process and then it is, you know there are variants or other problems associated with it, their solutions are very very useful for example you might have heard of this term hidden Markov model, these are used in speech processing, etc.

The notion of finite automaton of course used very heavily in formal verification of program and therefore this is something which is an extremely useful topic and a nice thing about this

finite automaton, that almost everything that we would like to do we can do. All the decision algorithms, all the decision problems they have algorithm and as we go to other classes, we will see more and more problems decision problems concerning such automaton will become, will be undecidable, that means we will not be able to write any program to do that, carry out those decision problems. But such problems are not there in general for regular languages and that is again another reason why it is so useful.