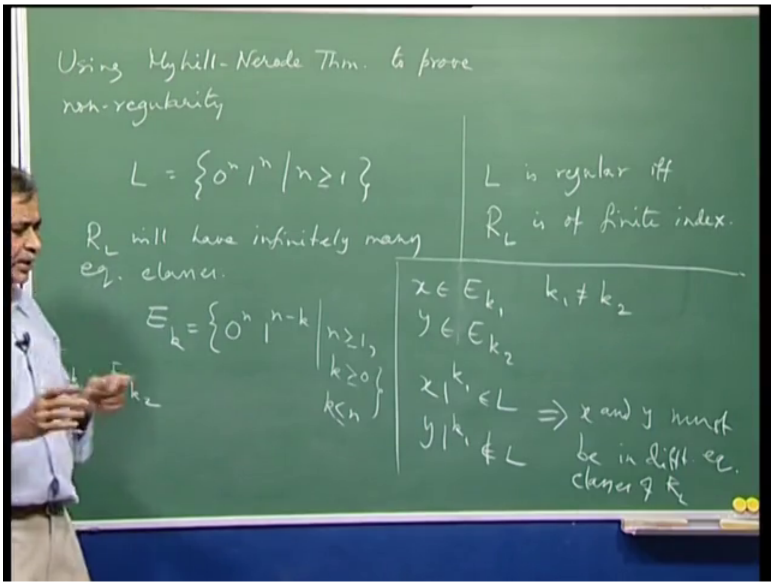
Theory of Computation.
Professor Somenath Biswas.
Department of Computer Science & Engineering.
Indian Institute of Technology, Kanpur.
Lecture-18.
Application of Myhill-Nerode Theorem.
DFA minimisation.

(Refer Slide Time: 0:21)



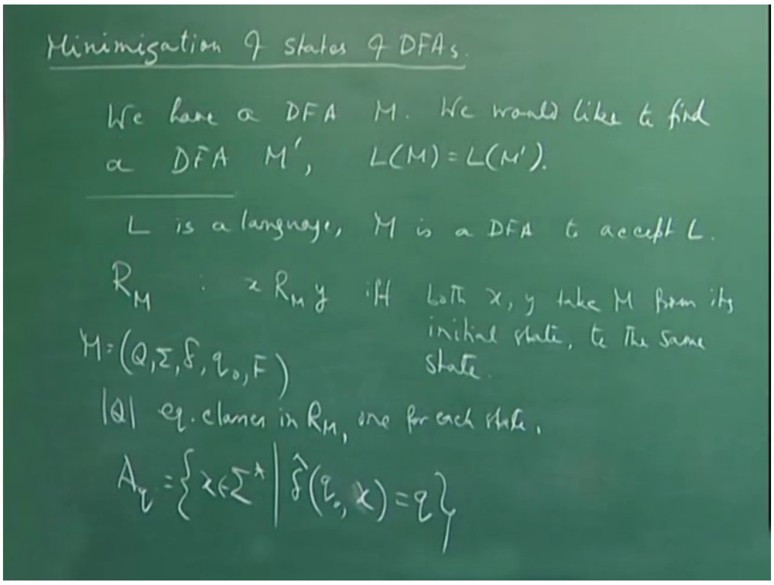
We can use myhill nerode theorem to prove that some languages not regular. So far we have used the pumping lemma, either the usual one or the generalised one to prove some language l to be nonregular. But now that we know myhill nerode here, this gives us another method to prove a language not regular. For example consider this language 0n, 1n, we know that this language is not regular, we prove its non-regularity by the pumping lemma. Now the way we use myhill nerode theorem to prove non-regularity is this. Recall we have one thing that myhill nerode theorem says is that l is regular if and only if rl is of finite index.

In other words the equivalence relation rl for a language l, if you consider that, then if the number of equivalence classes is infinite, that the language is not regular and if the language, number of equivalence classes is finite, then the language is regular. In this case it is fairly easy to show that rl for this language will have infinitely many equivalence classes. Let us consider sets of strings like this ek is 0n, 1n - k, and of course k is less than equal to n. Now take 2 strings from table 2 different eks. So let us say ek1 and e k2. I claim that a string, so let us write it here, let us say x is in e k1 and y is in e k2 and k1 is different from k2.

Consider this string x1 k1, so remember what is x, x will have some n number of zeros and some number of ones which is less than by k1 from the number of zeros. And if you add this 1, you know k1 once after x, then clearly this will have equal number of zeros and ones, therefore x1 to the power k1, this string will be in the language. Whereas if you take y from e k2, the deficiency, the number of ones it is less by is k2 and k2 is different from k1. So this will not be in the language. Right. So therefore from here we find that x and y, any string from e k1 and any string from e k2 must be in different equivalence classes of rl.

Then it is not difficult to see now, since we can have infinitely many ks, this makes it will have infinitely many sets like this, such that if you take 2 different such sets and if you take 2 strings one from here, one from here, they cannot be the same equivalence class, they must be in different equivalence classes. And therefore number of equivalence classes of rl for this language must be infinite. Therefore this proves that the language l is not regular. So this is a direct application of myhill nerode theorem to show a language to be not regular. Let us now consider this topic of minimisation of states of dfa.

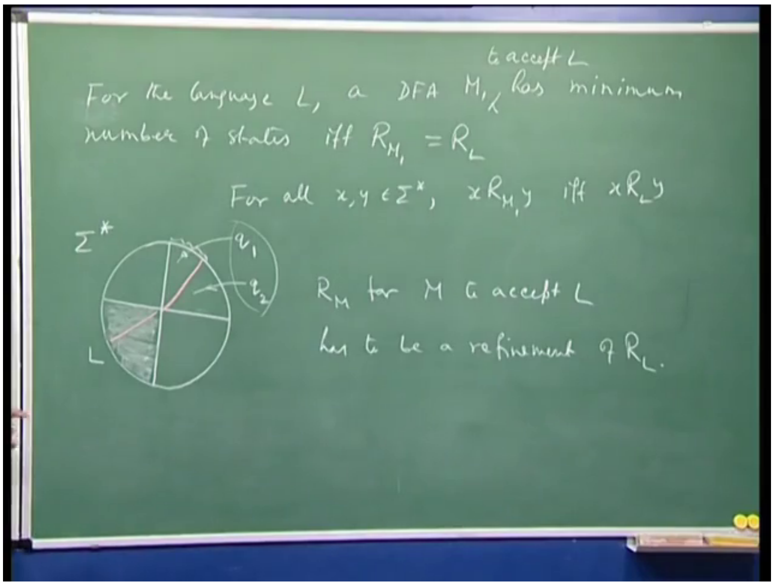
(Refer Slide Time: 6:41)



So what is the problem, we have a dfa m, we would like to find out, find the dfa m dash such that both m and m dash accept the same language and of all dfas accepting that language m dash has the minimum number of strings. Now the 1st thing we need to appreciate that even for this problem, myhill nerode theorem tells us a way of going about. Let us see how. So l is in let us say l is a language and m is a dfa to accept it. Right. Now we recall the definition of rm, what was rm, we said starting from this machine definition m, we said that 2 strings x and y are related by this, if and only if both x and y take m from its initial state to the same state. Okay.

Now we can write it in or the symbols we are familiar with. Suppose m is q, sigma, delta, q0 and f, then how many equivalence classes rm will have? Rm will have exactly so many equivalence classes and we remember that correspond into state q, this is small q, the equivalence class, let me, maybe i can call it as aq is the set of all strings x in sigma star such that delta hat q0, x is q, right. So what it means is sigma star is partitioned into equivalence classes and each equivalence class corresponds to one of the states of machine m. Now what is the best machine for this language l?

(Refer Slide Time: 11:21)



Now it is not difficult to see, if we go back to myhill nerode theorem, any machine for the language l, a machine dfa let us say m1 has minimum number of states, dfa m1 has minimum number of states, of course this dfa has to accept the language l, if and only if the equivalence relation corresponding to this machine rm 1 is exactly equivalence relation rl. In other words what we are trying to say is this that x for all, all x, y in sigma star, x rm 1 y, this holds if and only if x rl y holds. Why is so, because remember that any machine to accept l gives rise to of course an equivalence class rm and what we have proved in myhill nerode theorem that that rm is, what do we call, that it is, so let me get the picture. So let us say this is sigma star, right and let say for, we are just taking an example, rl is partitioned sigma star, let us see in these many equivalence classes. And remember the language l has to be one of these all a number of, union of some of these equivalence classes.

So for the purpose of illustration, let us write, let us say that this particular equivalence class, the strings in this, they constitute the language l, right, this is my l and of course the other strings are not in the language l. Now if we provide any machine, any dfa will accept l, myhill nerode theorem says that the corresponding rm for so let us say m to accept l has to be a refinement of rl. That means what, if you go back to the definition of refinement of 2 equivalent relations, the rm is an equivalence relation, rl is another equivalence relation. And we said that rm is a refinement of rl if the equivalence classes of rl are such that they are basically obtained by breaking one or more equivalence classes of rm.

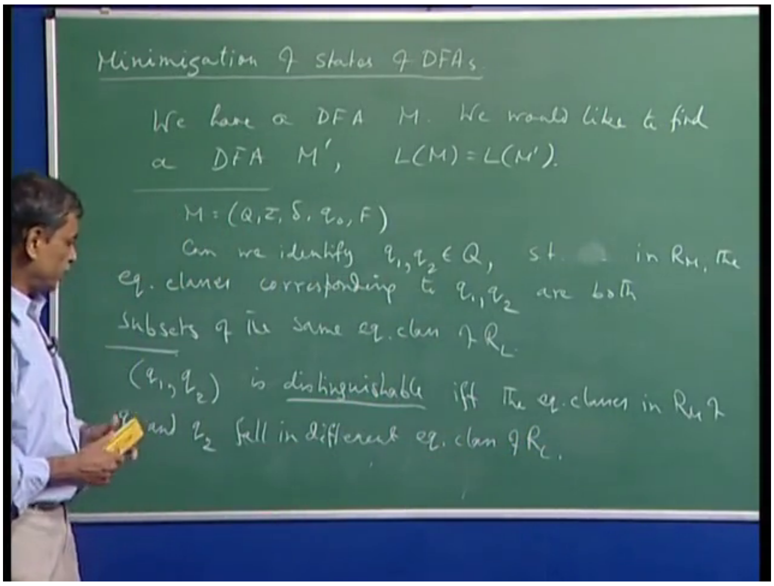
Actually i should have said is 0 or more because if m is your best machine then rm equivalence classes will be same as rl equivalence classes. So in other words if the machine m does not have the minimum number of states to accept the language, then it means that rm will essentially partition some of the equivalence classes, it will break up some of the equivalence classes. So let me show that situation in this manner. Let us say by this coloured thing, what we mean is now, previously rl had 1, 2, 3, 4 you know 1, 2, 3, 4 equivalence classes. Now this particular equivalence class of rl is partitioned into 2 and this is one equivalence class of rm, this is another equivalence class of rm and similarly here, maybe more things are broken up but this picture will suffice to make our point.

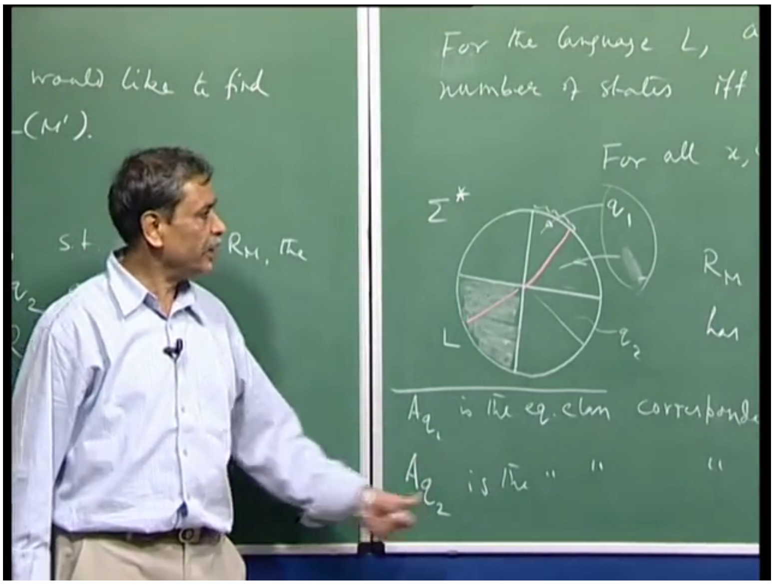
And this is the general understanding of refinement. Rm is a proper refinement of rl, that means it breaks up some of the old equivalence classes of rl for its equivalence classes. So what would minimisation mean? Minimisation would mean that right now i have 2 states, recall that for every state of the machine m we had an equivalence class in rl. If it is a proper refinement, rm is a proper refinement of rl, that would mean what, 2 or more equivalence classes corresponding to therefore 2 or more different states of the machine m, all of these actually fall in the same equivalence class of rl. Right.

So for example here, that maybe this particular equivalence class corresponded to state q1 and this particular, this particular equivalence class corresponded to the state q2 and what is their relation? Their relation is that the equivalence class in rm corresponded to the state q1 and the state q2, both these classes are subsets of the same equivalence class of rl. So if there is a way of combining these 2 states, then of course it would mean that this line would go this pink line would go and we will have, we will have improved in terms of the number of states in the dfa accepting the same language.

So this is why we are saying that for the language l, a dfa m1 to accept l has minimum number of states if and only if the equivalence classes corresponding to that machine rm 1, they are exactly the same equivalence classes of rl. If not, it would mean the rm 1 is a proper refinement of rl and therefore this kind of possibility of combining 2 equivalence classes, 2 of its equivalence classes into one is there. And thereby the possibility of reducing the number of states is there. Right, so now we can see what is our problem for minimisation.

(Refer Slide Time: 21:11)





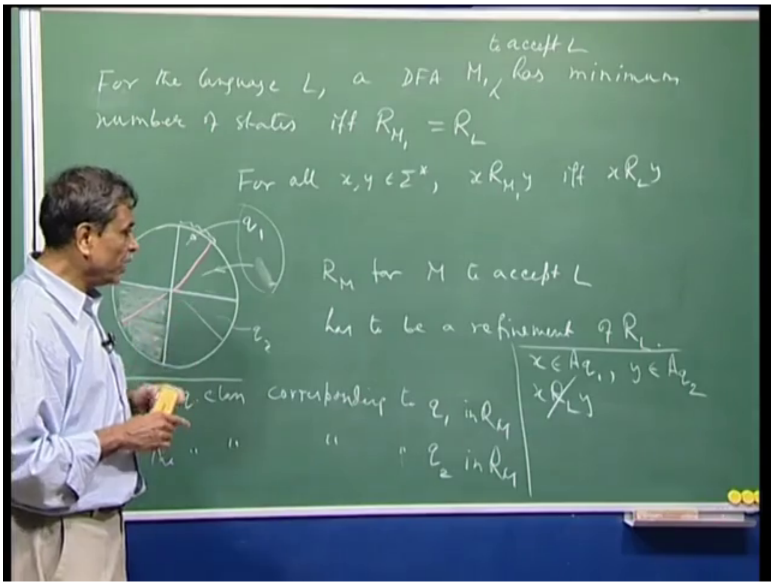
We are, we have been given a particular machine m, we have been given a dfa m, can i find 2 states repeatedly in this machine m such that the equivalence classes corresponding to these 2 states, equivalence classes for rm corresponding to these 2 states, both of these equivalence classes, they are parts of the same equivalence class of rl. How do identify such pairs of states? The question we are asking is for this machine m, let us say m is again as before q, sigma, delta, q0, f. The question we are asking is can we identify q1, q2 in q such that the equivalence classes corresponding to q1, q2 are both subsets of the same equivalence class of rl.

Now it might appear a little confusing because we are simultaneously talking about 2 different equivalence relations. When i say can we identify q1, q2 in q, such that the equivalence classes corresponding to q1, q2, now here we are talking of the relation, the equivalence relation rm. So let me write it clearly, such that, in let rm, the equivalence classes corresponding to q1 and q2 are both subsets of the same equivalence class of rl. Now so then you can see, then you can see the picture. So this, this is q1, this is q2, something like this and therefore at least intuitively we see that we can do something to merge these 2 states and that by we will improve in number of states.

Now what is easy, in fact what is more direct and equivalent of course to identify pairs of states which are not so. In other words let us look at this, what we are saying is that q1, q2, another way of saying this that we are trying to identify the spare q1, q2 is distinguishable, this is the term used to mean that the equivalence class in rm corresponding to q1 and the equivalence class corresponding to q2 in rm, these 2 equivalence classes are not subsets of the same equivalence class of rl. So iff the equivalence classes in rm of q1 and q2 fall in different equivalence class of rl. So this is the definition of this term distinguishable. I say pair of states is distinguishable if this is, this holds.

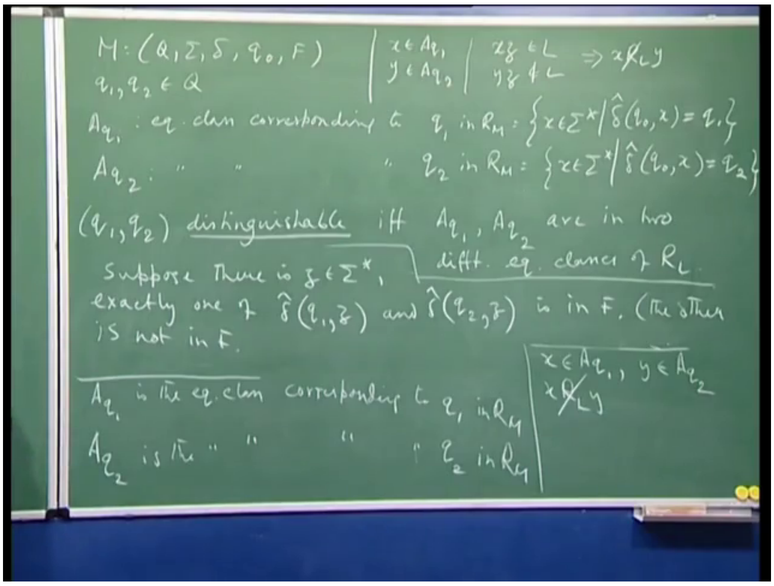
Now this holds when? That we know, see what do we know, when will 2, so let us see now that a q1 is the equivalence class corresponding to q1 in rm of course and a q2 is the equivalence class corresponding to q2 in rm. So when are they not in the same equivalence class of rl? So the situation is this is q1 and maybe not q2 here but if you take something like this, this was your q2, then this equivalence class and this is your a q1 and this is your a q2 and they are of course a 2 different equivalence classes of rl. And such pairs of states are easy to see or easy to figure out, at least we have a definition because they are in 2 different equivalence classes of rl.

(Refer Slide Time: 26:55)



So a string from here, let us say, what can i say about suppose that x is in a q1 and y is in a q2. Now suppose that x is not a equivalence, not in the same, it should be not related to y by the equivalence relation rl. That means equivalence class in which x is there and in which y is there, these 2 classes fall in different equivalence classes of rl. And that is what was this question, so that is how to answer. Right. And what would, what do i know about a q1 and a q2? A q1 all those strings which take m from q0 to the state q1 and a q2 are all those strings which take the machine m from q0 the initial state to the state q2. Right.

(Refer Slide Time: 28:11)



Now let us write this in terms of the machine behaviour. So let us summarise what we are saying, this is a machine m, q1 and q2 are the 2 states in the machine and equivalence class, a q1 is the equivalence class corresponding to the state q1 for the equivalence relation r rm and that by definition is this, we have seen the definition of rm is the set of all strings which take the machine from its initial state to q1 and similarly for q2. Now what we are saying that these 2 states are distinguishable if and only if these 2 equivalence classes are in 2 different equivalence classes of rl. And how can i figure that out? Now look at this way, that suppose there is a z, some string z, such that delta hat, let us say exactly one of delta hat q1 z and delta hat of q2z.

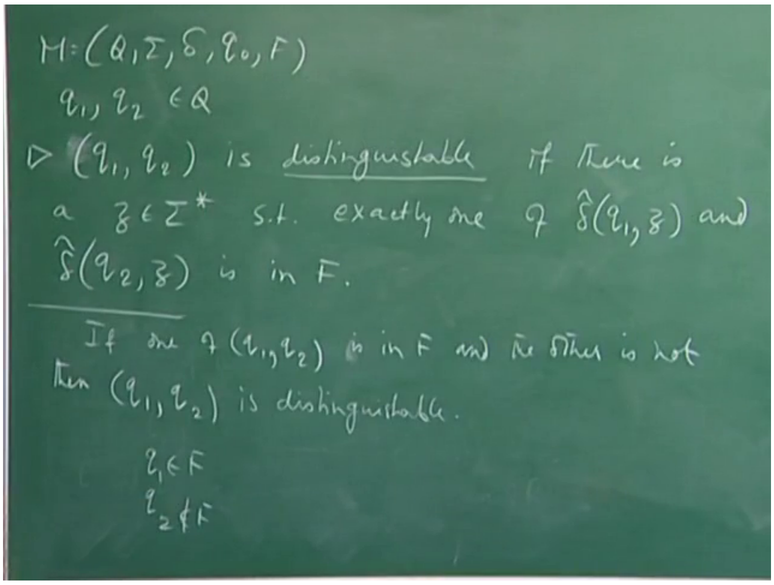
Remember these 2 will be 2 states, what we are saying that suppose there is some string z such that exactly one of these 2 states is in f, basically that means the other is not, other is not in f. Consider this equation, that there is some string z such that delta hat of q1, z, let us it is one of the final states and delta hat of q2, z is not a final state. But what does it mean? Let us say string x took the machine as we said, suppose from q0 pqr and now consider this z. So now let us say, so suppose x, this string x is in a q1 and y is in a q2, right, suppose this is the situation. Then now look at these 2 strings xz and yz.

So of course x is in the equivalence class qi and now you apply this string z as a concatenated string to x on x, z similarly you did for yz. And now what do you find that totally if you take this string xz, x took the machine m from q0 to q1 and then from q1 z to k2 let us see final state and whereas y took the machine from its initial state to q2 and then the same z took the machine to a non-final state. That means what? That means this string will be the language and this string will not be in the language. That means what, that the 2 strings x and y is not related by the relation equivalence relation rl and therefore of course it means that a q1 and a q2 are in 2 different equivalence classes of rl.

And that is why this is the situation in which we can see q1 and q2, this pair of states is distinct visual. And coming back to what we were saying earlier, it means that there is no way we can merge these 2 states and hope to get dfa which will accepting the same language. Right. However you see that this is now at least it seems that given a pair of states, look at the situation what we are saying that suppose there is a string z in sigma star, exactly one of these 2 states is a final state, this is something we can test, as we shall see by means of an algorithm. And thereby we will be able to figure out distinguishable states.

And from what i know from myhill nerode theorem that such pairs of distinguishable states if q1 and q2 are distinguishable, then they cannot go into the same equivalence class of rl and my the best dfa or the dfa that will correspond 2 whose rm will correspond to rl, that will be by somehow combining those states which are not distinguishable. So let us now spend time to find out how we can test this conveniently.

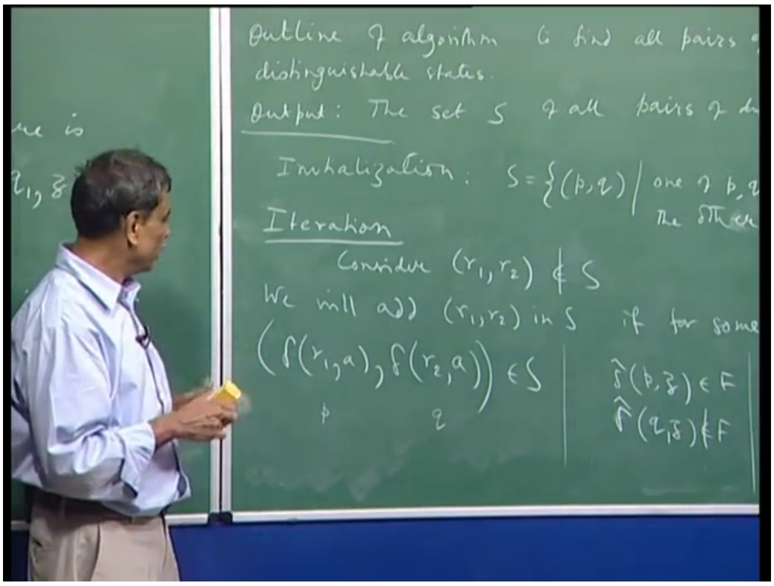
(Refer Slide Time: 34:35)

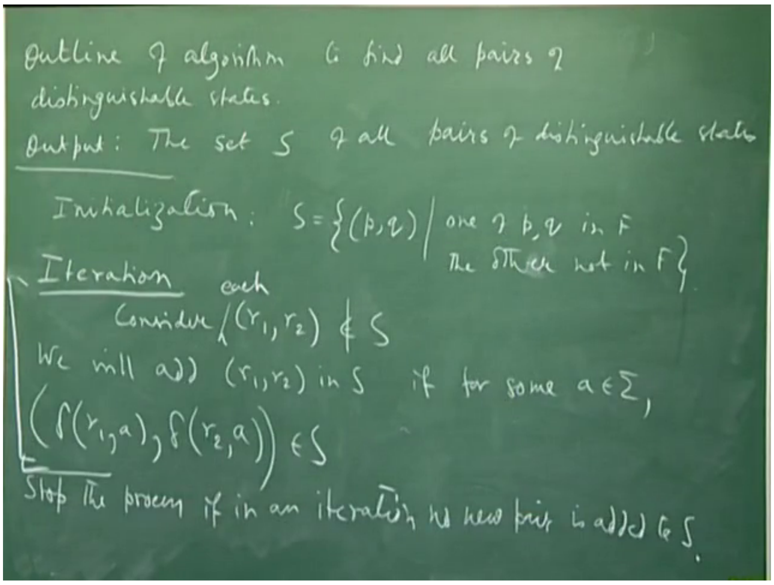


So we have motivated ourselves towards this definition of distinguishability. What we are saying is that a pair of states is distinguishable if we can find a string z such that exactly one of delta hat q1z, this is one state and this is the other states going from q2z using z. Exactly one of these states is in f. In such a case if we can find this condition to be true for a pair of states, we will see that pair of states is distinguishable. And there is a systematic way of finding all pairs of distinguishable states. 1st of all one thing is very clear that if one of q1, q2 is in f and the other is not, then q1, q2 is distinguishable. Why so? Because in that case, in this situation, what is the situation, let us say q1 is in f, one of the final states and q 2 is not in f.

Then this pair is distinguishable. That is because consider z to be epsilon, the empty string. Delta hat q 1 z, q1 epsilon is q1 which is a final state, whereas delta hat of q2z, z is epsilon, this will then be, when z is epsilon, it will be q2 and q2 is not final state or stop in that case of course this pair is distinguishable. So that is how our algorithm would start to find out the set of all possible pairs of distinguishable, all possible distinguishable path of states.

(Refer Slide Time: 37:22)





An outline of algorithm to find all pairs of distinguishable states, okay. So the initialisation will start from this observation. So let us see what is desired output. Output should be the set of, the set s let say of all pairs of distinguishable states. So this is the set that i would like to construct given a machine m. So initialisation for s we will, we will build s iteratively. Initialisation is from this observation that f is all p, q such that one of p or q in f and the other is not. So i have a 1st approximation to my set of all distinguishable states. Now let us see how we can starting from this initialisation how we can augment this set iteratively.

So this is the iteration part, consider r1, r2 which a pair of states which is not already in s and we will add r1, r2 in s. So we will add, we will augment our set s by this new pair if for some a in sigma, delta r 1, a and delta r2, a, this was in s. In other words what we our iteration is, considering all pairs r1, r2 which are not already there in s, this pair r1, r2 will be, we will put it, put the spare also in s, provided i find something a, such that, you know this pair was always there in s. What is the justification? Suppose delta r1 is p and delta r2 is q, which means that p, q was already in s and therefore they are distinguishable.

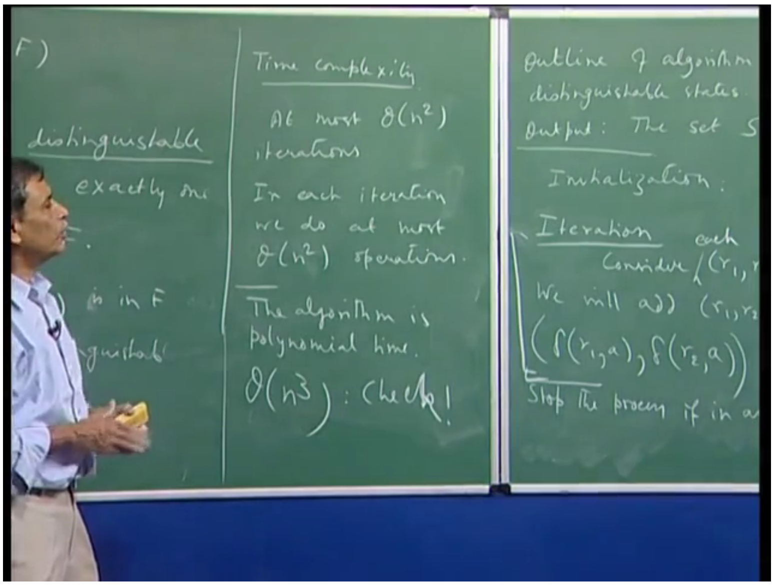
So there is a string z such that delta hat of p, z let say in one of the final states is one of the final states and delta hat of q, z is not. And now consider the string a z, a takes r1 to p and rest of the string z takes p to a final state. Whereas the same string a z, what its behaviour with respect to r2, r2 goes to q on this symbol a and from q this string z takes the machine to a statement is not final state. So here is a string az which is such that exactly one of this will take r1 of the of the 2 states r1 and r2, a z is a string such that it will take the machine to a final state from 1, exactly one of these states r1 r2. And therefore r1, r2 is distinguishable. So this is the justification, let me rub this out the justification part and our algorithm therefore will mean that in the iteration part you are considering all pairs of states which are not already in s and such a pair you will add to s.

You know if you find some symbol a, remember that number of symbols in sigma is finite, so one of the another u test this condition. You have chosen a pair r1, r2 which is not in s, consider delta r1, a and delta r2, a, this pair ask is it already in s, if it is in s, then put this pair also in s and so on. And finally we must see when to stop, we say that stop the process if in an iteration no new pair is added to s. Okay. So remember the iteration is this entire thing, so maybe just emphasises i should write consider each r1, r2 not in s, so in one iteration, remember in an iteration you already have a set of pairs which is in s and some of the pairs which are not in s, these are the set of all pairs of states.

So in an iteration, in one iteration you will consider all the elements which are not in s and as you consider one pair, you may find this condition to be true. So in that case that there will be added and so on. Now imagine an iteration where all the pairs which are not in s, you tested this condition and you could not put this any new pair, that pair to s, that was true for all the pairs not in s, in that case we see that iteration did not add any new pair to s and in such a case we will stop the algorithm. What is the time complexity of this algorithm? Is, it is fairly easy to see that this is a polynomial time algorithm because in each iteration how many total number of pairs states are there?

If there are n states, there are you know n square pairs of the order n square pair, actually n q2 pairs and, actually we should talk of an ordered pair, right because these are, if pq is distinguishable, then qp is also distinguishable but that is the detail, the main thing is in each iteration we add at least one new pair. So how many total number of iterations? So let me write here the time complexity of the algorithm and then we will bother about the correctness.

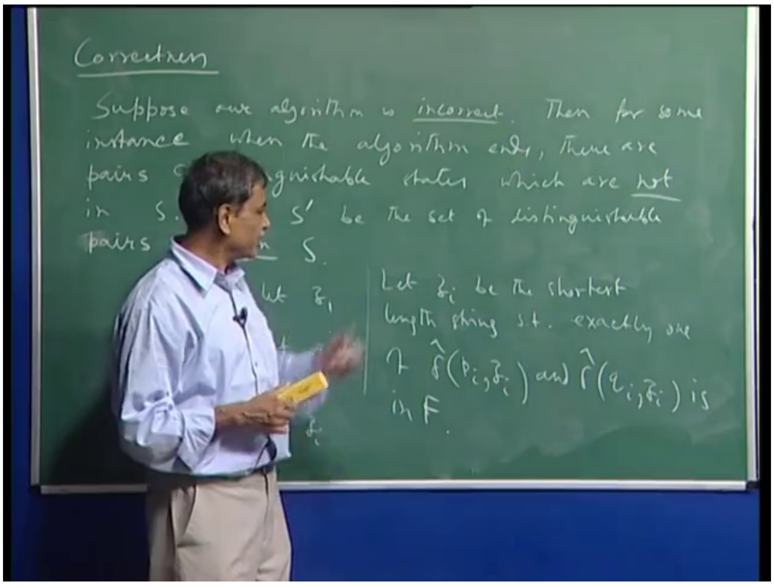
(Refer Slide Time: 46:57)



Time complexity, there are at most order n square iterations. And what do you do in each iteration, in each iteration you consider all the pairs which are not in s and there can be order n such pairs which are not in s, sorry order n square such pairs not in s, right. And you do something, you do this test for each such pairs. So you can see that in each iteration we do at most order n square operations. So therefore the algorithm is polynomial time. But this is what i said is kind of pessimistic, right. You should be able to see that the algorithm really, the way i have said as if it is order n 4 but actually it is order n cube.

Just to make this point that it is polynomial time i made this rather loose argument but you should be able to see this that this is actually order n cube. Though from what i have said, it does not immediately follow, you should be able to do this, check, check that that this is indeed order n cube. And now let us talk about the correctness of the algorithm.

(Refer Slide Time: 49:21)

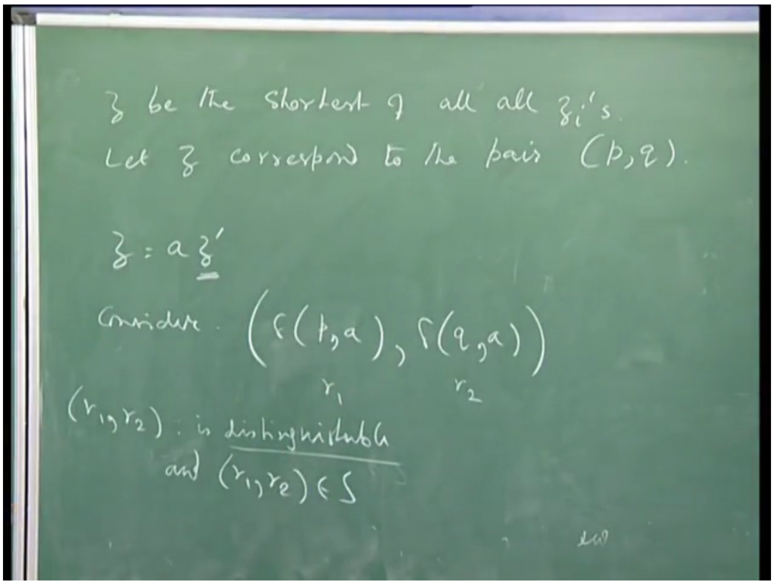


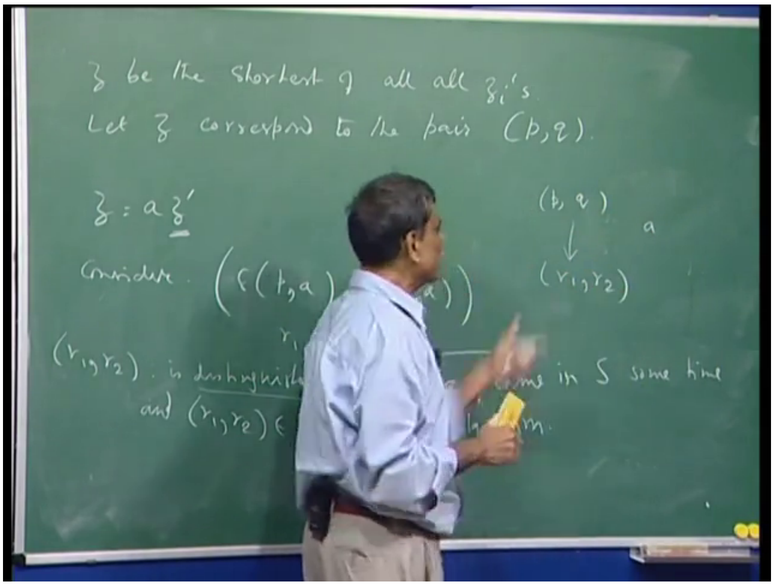
So let us see why they allow them that we have outlined is indeed correct. When is it correct? If when we stop, by then we would have discovered states which are distinguishable. Right. We end with a certain set s and if that s contains every pair of distinguishable states, then of course the algorithm is correct. I would like to show that is indeed true and let us prove this by contradiction. Suppose our algorithm is not correct, then suppose, not suppose, then there must exist, then for some instance and here the instance is of course you are given a dfa, then for some instance when the algorithm ends, there are pairs of states, pairs of distinguishable states which are not in s that is why the algorithm is incorrect.

The incorrect means this condition must be true, there must be some instance where it is working incorrectly, that means it has not been able to complete the entire set of, it has not been able to put all the set of distinguishable states into distinguishable pairs of states in s. Right. Now, let s prime be the set of distinguishable states, pairs of states not in s. So these are s prime are all those pairs which auto have been in s but they are not in s. So supposing these are these pairs p1, p2 or let us say p1, q1, p2, q2 and so on. And now suppose for each one let say we see for each pi, qi, each such distinguishable pair which is not in s, let zi, let z1 so in this case zi.

So these are strings and what are these strings, they distinguish these pairs, these are the shortest. So let zi be the shortest length string, such that exactly one of pi, zi and qi, zi is in s, right. So that means these strings zi witnesses the fact that the pair pi, qi is distinguishable. And we are saying that zi be the shortest length string of all such pairs, let of all such strings, there will be infinitely many strings, let zi be the shortest. Now let z be the shortest of all these strings. So in each zi is the shortest witness for that corresponding pair and of all these, let particular z, z be the shortest of all zi's.

(Refer Slide Time: 57:53)





And let z correspond to the pair p, q. Right. So now let the string z be some a followed by let say z dash. That is the, the string z begins with the symbol a, right. Now, consider this pair of states delta p, a and delta q, a. Now let this be the state r1 and let this be the state r2. Now i claim that r1, r2 is of course, is firstly 2 things i claim, is distinguishable and r1, r2 is in s. That means r1, r2, this pair, this distinguishable pair would happen discovered by our algorithm. Why, the answer is, the reason for that is very simple, because the fact that r1, r2, they are distinguishable, that is witnessed by z prime because if you go from r1 on z prime, let us say this from p1, a goes to r1 and r1 on z prime goes to let say final state and this does not.

So r1 and r2 will be distinguishable and the shorter string, shorter than z axis, so in that case that would have been the pair r1, r2 i would have picked up and that prime would have been the z and not this. So this is true, r1, r2, this pair this distinguishable of course, that is easy to see and what more, this pair is already in s. And now comes the contradiction that, consider, so since r1, r2 are in s, that means this pair r1, r2, this pair r1, r2 was included, came in s sometime in the algorithm, maybe in the initial phase or maybe in one of the iterations.

So now consider iteration immediately following that when r1, r2 came into s. By then the pair p, q would not have been in s, in any case p, q was never in s, pq was based by our algorithm but when you take the pair pq and consider the symbol a, then you would have got the pair r1, r2 which was already there in the set s and then pq would have been declared as distinguishable. So therefore we have a contradiction and therefore this is not true that there is some pair of distinguishable states which our algorithm had failed to discover.