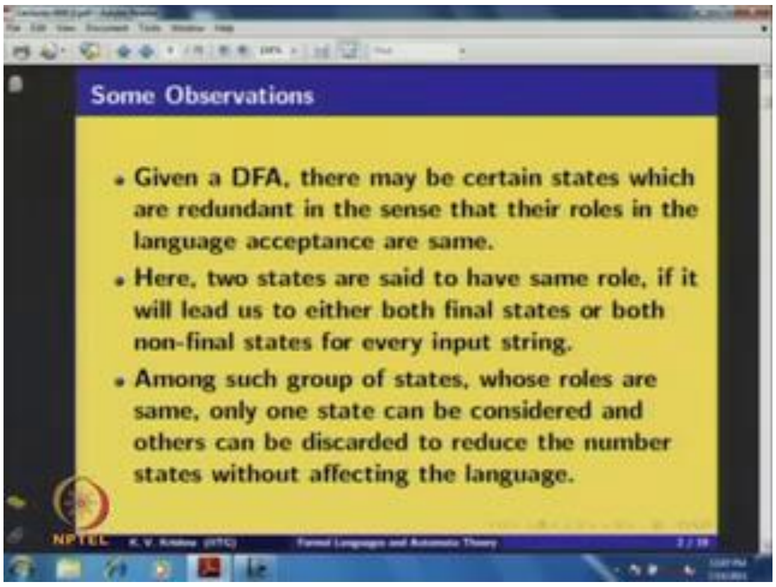
Formal Languages and Automata Theory

Prof. Dr. K. V. Krishna
Department of Mathematics
Indian Institute of Technology, Guwahati

Module - 4
Minimization of Finite Automata
Lecture - 2
Minimization

In the process of Minimization of a DFA in my previous lecture, I have introduced characterization of languages that are accepted by DFA, so called Myhill–Nerode theorem, so we will be using that to minimize the number of states in a DFA. As you recollect, when we are converting an NFA to DFA we are getting so many states and we are seeing that there are certain states whose roles are same. And thus there may be a process required to reduce those resonant states, first let us see these observation in this direction.

(Refer Slide Time: 01:10)



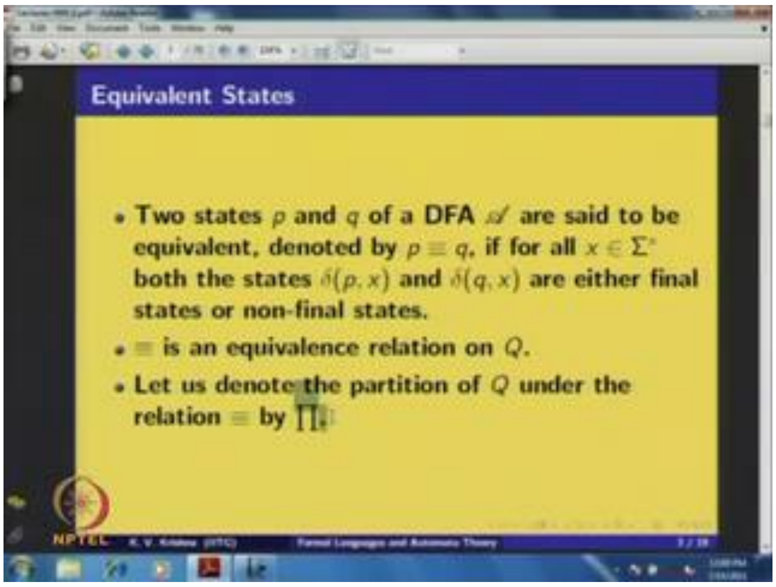
Some Observations

- Given a DFA, there may be certain states which are redundant in the sense that their roles in the language acceptance are same.
- Here, two states are said to have same role, if it will lead us to either both final states or both non-final states for every input string.
- Among such group of states, whose roles are same, only one state can be considered and others can be discarded to reduce the number states without affecting the language.

So, given a DFA there may be certain states, which are redundant in the sense that their roles in language acceptance are same, that is a point number 1. Here, two states are set to have same role, if it will lead us either in both final state or both non final states for every input string. Well look at here, we said two states have the same role, because here DFA use for the language acceptance; that means, if you put any string in the initial state, if you are reach into the final state we say it is accepted.

Now, two states are set to be equivalent here, if you are putting any string in both the states, in both are reaching to final states or both are reaching to non-final state, we say that they their roles are same. We will use this intuition to formulize and introduce the notion of equivalence of states. So once we can find such an equivalent states, then what we do among the group which are equivalent and we will remove all other states; and only one state if you pick up and you can construct a DFA with that a redundant states.

(Refer Slide Time: 02:33)



Equivalent States

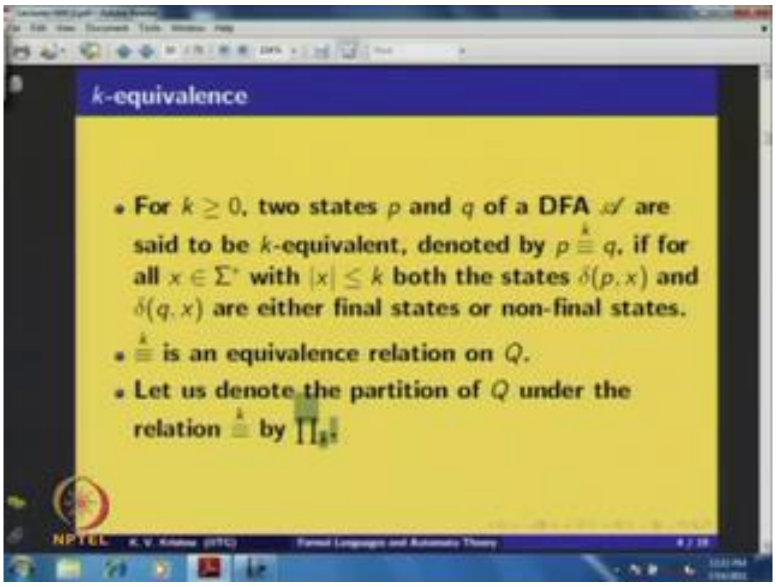
- Two states p and q of a DFA $\mathscr{A}$ are said to be equivalent, denoted by $p \equiv q$, if for all $x \in \Sigma^*$ both the states $\delta(p, x)$ and $\delta(q, x)$ are either final states or non-final states.
- $\equiv$ is an equivalence relation on Q.
- Let us denote the partition of Q under the relation $\equiv$ by $\prod$.

So, in that process let us formally define the notion equivalent of states, now two states p and q of a DFA, A are set to be equivalent denoted by p equivalent to q here, I am calling it equivalent anyways. If for all x and sigma star, both the states delta of p comma x and delta of q comma x are either final states are non-final states that means, if you put any string in both the states, you both are reaching to final states or both are reaching to non final states we say they are equivalent.

Formally let us introduce the equivalence between states like this and we can clearly observe that this is an equivalence relation on the states set q, you can observe that it is reflexive. Because if you put any string in a particular state delta of p comma x and delta of p comma x as it is same, both are final states or both are non-final states. So, that way it is reflexive and you can observe the symmetry very clearly and transitivity, so that you can understand that this is an equivalence relation on the straight set q.

Now, once you have an equivalence relation on a particular set, we will get a partition of that set, let us denote the partition corresponding to this equivalence relation by pi.

(Refer Slide Time: 03:59)



k-equivalence

- For $k \geq 0$, two states p and q of a DFA $\mathscr{A}$ are said to be k-equivalent, denoted by $p \overset{k}{\equiv} q$, if for all $x \in \Sigma^*$ with $|x| \leq k$ both the states $\delta(p, x)$ and $\delta(q, x)$ are either final states or non-final states.
- $\overset{k}{\equiv}$ is an equivalence relation on Q.
- Let us denote the partition of Q under the relation $\overset{k}{\equiv}$ by $\prod_k$.

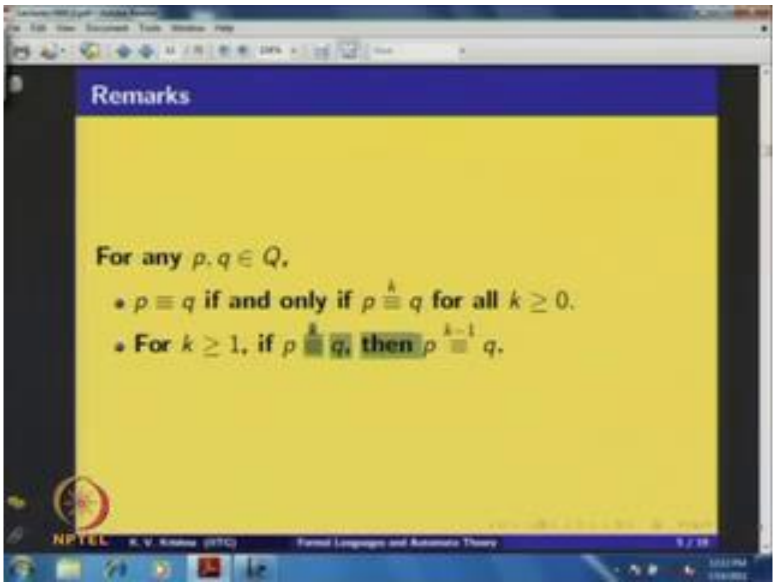
Now, you understand that evaluating this equivalence relation on a state set it is not that easy, because to understand two states are equivalent, you have to put every string from sigma star and see whether both are resultant states or final states are non-final states. So, since there are countably infinite many string in sigma star, it is not very practical to evaluate the equivalence directly that means, by applying every string in both the states and seeing whether both are final states or non-final states.

So, in that context we will in a more practical direction, we will first introduce a k equivalence, where we put some bound on the number of strings that you will be testing for the purpose. So, that we call it is k equivalence as shown here, we define for k greater than equal to 0, two states p and q of a DFA A are set to be k equivalent denoted by p k equivalent to q I call it as k equivalent to q.

If for all x whose length is less than or equal to k, both the states delta of p comma x and delta of q comma x are either final states or non-final states that means, we are putting the restriction on the length of the strings. That means, up to k length including k the strings you apply in both the states and see what are the resultant states? If both the resultant states are final states, then you say they are k equivalent.

If both the resultant states are non-final states, then say they are k equivalent, otherwise that means if one is a final state and other is a non-final state, then you say they are not k equivalent. Again you can understand that this k equivalence relation that is an equivalence relation on the straight set and the corresponding partition, let us denoted by pi k; so using this k equivalence we can now approach to the state equivalence.

(Refer Slide Time: 06:05)



Remarks

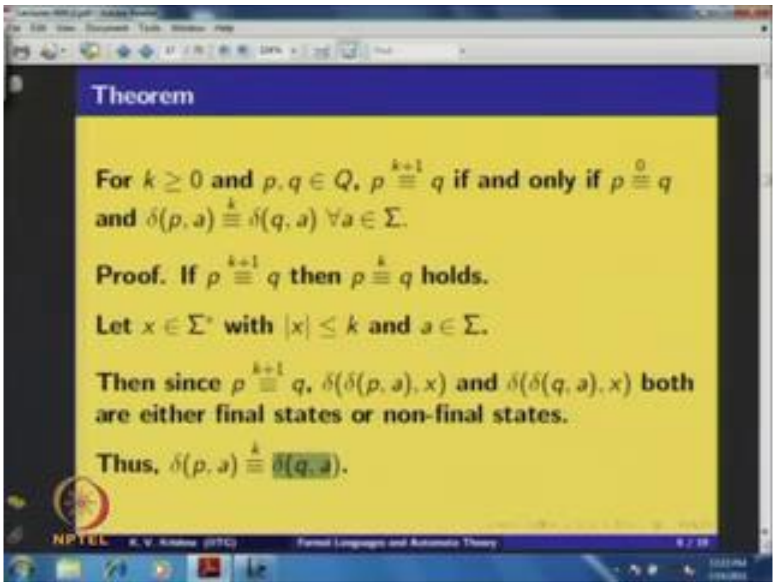
For any $p, q \in Q$,

- $p \equiv q$ if and only if $p \overset{k}{\equiv} q$ for all $k \geq 0$.
- For $k \geq 1$, if $p \overset{k}{\equiv} q$, then $p \overset{k-1}{\equiv} q$.

So, here are the remarks you can quickly understand that, if two states are equivalent p is equivalent to q, if and only if p is k equivalent to q for all k greater than equal to 0. Because, for all length strings you will be submitting in both the states p and q and see the resultant states, if they are essentially k equivalent, then we can observe that they are equivalent.

And this is a another quick observation, for k greater than equal to 1, if two states are k equivalent then they are k minus 1 equivalent, the reason we are submitting the strings of length up to k in both the states p and q. And we are seeing that both are final states or non-final states, then you assume p is k equivalent to q. That means, since we are submitting the strings of length up to k, we are submitting any where length strings up to k minus 1 also that way it is clear that, if p and q are k equivalent, then they are k minus 1 equivalent also.

(Refer Slide Time: 07:04)



Theorem

For $k \geq 0$ and $p, q \in Q$, $p \overset{k+1}{\equiv} q$ if and only if $p \overset{0}{\equiv} q$ and $\delta(p, a) \overset{k}{\equiv} \delta(q, a)$ $\forall a \in \Sigma$.

Proof. If $p \overset{k+1}{\equiv} q$ then $p \overset{k}{\equiv} q$ holds.

Let $x \in \Sigma^*$ with $|x| \leq k$ and $a \in \Sigma$.

Then since $p \overset{k+1}{\equiv} q$, $\delta(\delta(p, a), x)$ and $\delta(\delta(q, a), x)$ both are either final states or non-final states.

Thus, $\delta(p, a) \overset{k}{\equiv} \delta(q, a)$.

Now, let us try to understand that, what is the relation, because here we give a remark that p is equivalent to q if and only if for all k, p is k equivalent to q that means, for all k we have to test, but is there a way to understand that. Because, again here this is an infinite test, because for all k greater than equal to 0 we have to test, so in this direction of finding an appropriate way to understand that at a finite stage let us first to have the following results.

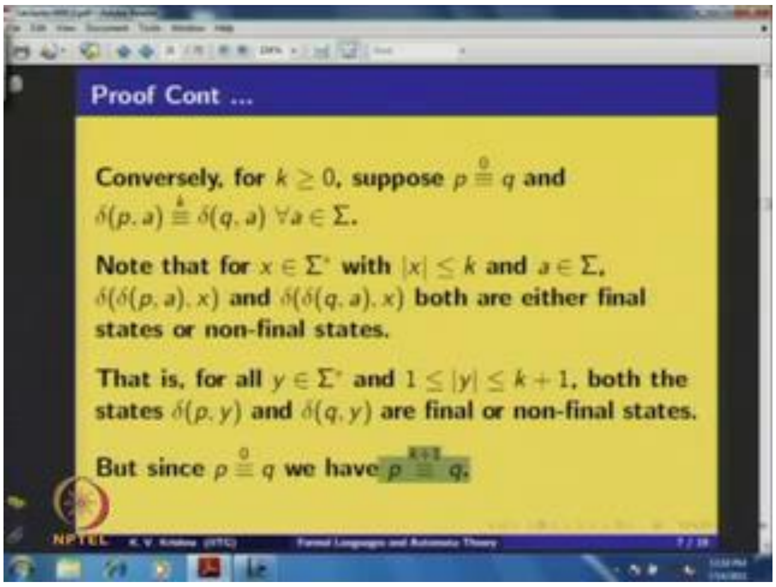
Now, the theorem here is for k greater equal to 0 and if you take any two states p and q in, now p and q are k plus 1 equivalent, if and only if there are zero equivalent and delta of p, a and delta of q, a are k equivalent for all A in sigma. So, look at this theorem, here we are characterizing the k plus 1 equivalence in terms of the lower equivalence is here k equivalent of course, here it is 0 degree equivalence that we are looking at.

So, once we understand this, we will be able to evaluate from this criterion that, if you have understanding about k equivalence, then one more step about k plus 1 equivalence this criteria will be helpful, so let us look at the proof of this first. If p is k plus 1 equivalent to q, then as we have remark p and q or k equivalent and thus you can understand that, from k plus 1 equivalent to k equivalent and k equivalent to k minus 1 equivalent and so on.

You can pour down to zero equivalent and you can clearly observe that, if p and q are k plus 1 equivalent, then p and q zero equivalent anyway. So, from this statement I can conclude that p and q are zero equivalent and now to understand delta of p, a and delta q, a are k equivalent for all a, to understand that let x and sigma star of length less than or equal to k and you will pick an arbitrary a in sigma. Since, p and q are k plus 1 equivalent, we can un understand that delta of delta p comma a x, then the yet the total length is k plus 1.

So, the resultant state when you apply in p, the resultant state when you apply in q the length of string here is a x is k plus 1, both the states either final states or non final states, because from the hypothesis p is k plus 1 equivalent to q, here we are applying the strings of lengths k plus 1. And thus you can understand that these states, the resultant states then you apply the string a x in p and q both are either final states or non final states. And from this you can quickly conclude, then that since this is true for all x of length less than equal to k delta of p, a and delta of q, a are k equivalent.

(Refer Slide Time: 10:29)



Proof Cont ...

Conversely, for $k \geq 0$, suppose $p \overset{0}{\equiv} q$ and $\delta(p,a) \overset{k}{\equiv} \delta(q,a)\ \forall a \in \Sigma$.

Note that for $x \in \Sigma^*$ with $|x| \leq k$ and $a \in \Sigma$, $\delta(\delta(p,a),x)$ and $\delta(\delta(q,a),x)$ both are either final states or non-final states.

That is, for all $y \in \Sigma^*$ and $1 \leq |y| \leq k+1$, both the states $\delta(p,y)$ and $\delta(q,y)$ are final or non-final states.

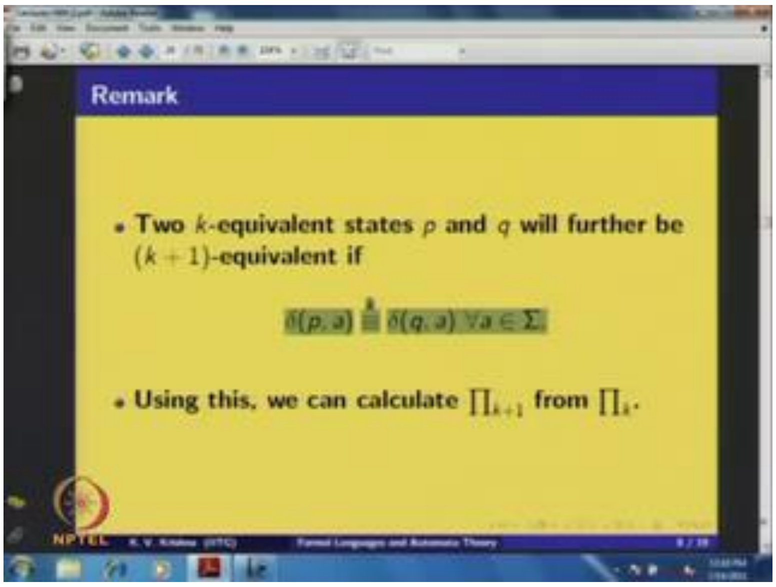
But since $p \overset{0}{\equiv} q$ we have $p \overset{k+1}{\equiv} q$.

Conversely for k greater than equal to 0, suppose p is 0 equivalent to q and delta of p, a is k equivalent to delta of q, a for all a in sigma, now what we have to prove, we have understand that p and q are k plus 1 equivalent. Now, since we are assuming that delta of p, a and delta q, a are k equivalent for all x of length less than equal to k, and for any a in sigma delta of p, a x and delta of q, a x, this two states both are final states or non final states.

That means, from this what we can understand that, because here is restriction delta of p, a in this state you are applying the string of length less than equal to k and delta of q, a in this also you are applying a string of length less than equal to k. You can understand that, for if this with respect to p and q, delta of p, y, delta of q, y the resultant states when you apply any string of length between 1 and k plus 1, because minimum 1 is here and of length up to k you are adding.

That means, of length up to k plus 1 that means, length between 1 and k plus 1, any string y if you apply in both p and q, they are final states or non-final states and in the hypothesis it is given that p is 0 equivalent to q. So, now what we have p and q are zero equivalent and if you apply any string of length varies from 1 to k plus 1 the resultant states are both final states or non final states. And thus what you have is in p and q, if you apply any string of length up to k plus 1 both are final states or non final states and hence, we have the conclusion p is k plus 1 equivalent to q. So, now the criteria here, what we have to conclude that two states are k plus 1 equivalent if and only if, they are zero equivalent and for all a, if you apply in both the states, if they are k equivalent, then this is a criterion, this is a characterization for p is k plus 1 equivalent to q.

(Refer Slide Time: 12:52)



Remark

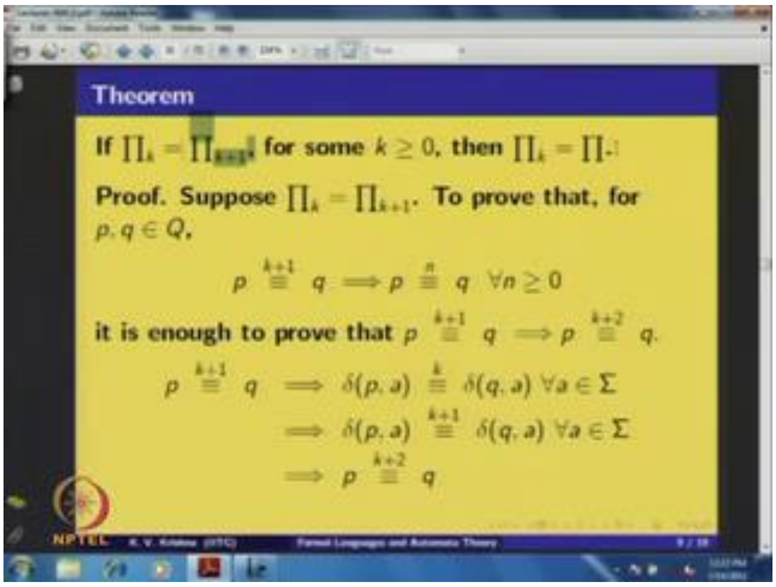
- Two k-equivalent states p and q will further be $(k+1)$-equivalent if

$$\delta(p,a) \overset{k}{\equiv} \delta(q,a) \ \forall a \in \Sigma.$$

- Using this, we can calculate $\prod_{k+1}$ from $\prod_k$.

So, two k equivalent states p and q will further be k plus 1 equivalent, if delta of p, a is k equivalent to delta q, a for all a in sigma, so this is the criterion to evaluate k plus 1 equivalence from k equivalence. And using this now I can say that, we can calculate the partition by k plus 1 from pi k that means, from k equivalence we can, how to evaluate k plus equivalence this criteria, this criterion is helping us.

(Refer Slide Time: 13:28)



Theorem

If $\prod_k = \prod_{k+1}$ for some $k \geq 0$, then $\prod_k = \prod$.

Proof. Suppose $\prod_k = \prod_{k+1}$. To prove that, for $p, q \in Q$,

$$p \overset{k+1}{\equiv} q \Longrightarrow p \overset{n}{\equiv} q \ \forall n \geq 0$$

it is enough to prove that $p \overset{k+1}{\equiv} q \Longrightarrow p \overset{k+2}{\equiv} q$.

$$\begin{aligned} p \overset{k+1}{\equiv} q &\Longrightarrow \delta(p, a) \overset{k}{\equiv} \delta(q, a) \ \forall a \in \Sigma \\ &\Longrightarrow \delta(p, a) \overset{k+1}{\equiv} \delta(q, a) \ \forall a \in \Sigma \\ &\Longrightarrow p \overset{k+2}{\equiv} q \end{aligned}$$

Now, as I had mention that k equivalence, for all k greater than equal to 0 is essentially equivalent to the state equivalence what we have defined now, but since it is an infinite, there are infinitely anything many things that we have to test. Now, this theorem will be helpful that if you have obtain at a particular stage pi k is pi k plus 1 for some k greater than equal to 0, then we can conclude that, that particular partition pi k is in fact, the partition corresponding to the state equivalence pi.

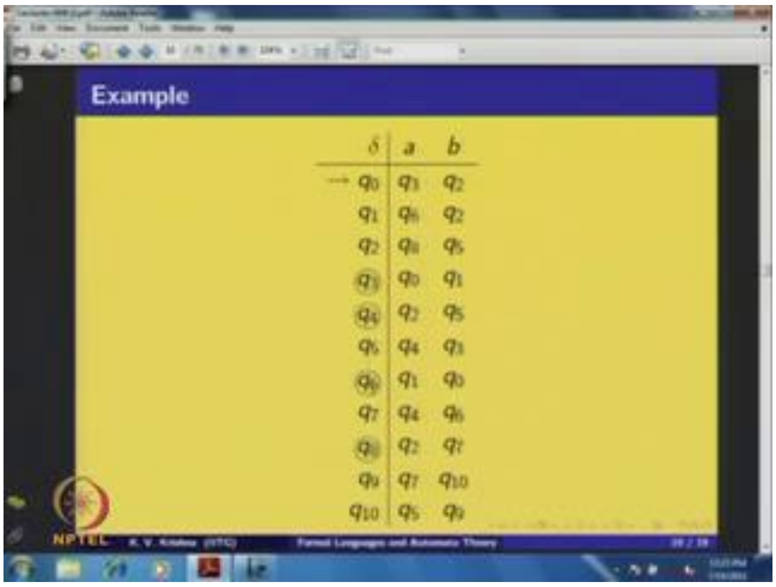
So, this theorem will be helpful to understand that, we can actually have a finite process to understand the straight equivalence, so let us first prove this theorem. Suppose, pi k is equal pi k plus 1 for some k greater than equal to 0, to prove for p and q and q, k plus 1 equivalent implies, there k plus 1 equivalent implies they are equivalent for all a greater than equal to 0. It is enough to prove that if there k plus 1 equivalent, then there k plus 2 equivalent also, once I prove this then by induction from k plus 2 to k plus 3, we can go and k plus 3 to k plus 4 we can go and so on.

Thus we can understand that, if p and q are k plus 1 equivalent, then p and q are n equivalent for all n greater than equal to 0, so we just proved that p and q are k plus 1 equivalent implies there k plus 2 equivalent, now let us consider that. Assume p and q are k plus 1 equivalent that implies by definition delta of, if you apply any string, as we have proved this criterion if p and q are k plus 1 equivalent. Then you can understand that delta of p, a and delta of q, a are k equivalent, for all a belongs to sigma just we have proved this criterion.

And now what do we understand this implies delta of p, a and delta of q, a are k plus 1 equivalent for all a belongs to sigma. Now, once we have this criterion, then we can conclude that p and q are k plus 2 equivalent, once again we see that once you start with p and q are k plus 1 equivalent, then clearly for all a belongs to sigma we have this. Now, this is a criteria to understand that delta of p, a is k plus 1equivalent delta of q, a and once again using that theorem we can conclude that p and q are k plus 2 equivalent,

And hence, by induction two states are k plus 1equivalent implies there n equivalent for all n greater than equal to 0 and hence, pi k is equal to pi k plus 1, for some k greater than equal to 0, then that pi k is actually the partition is corresponding to the partition on state equivalence.

(Refer Slide Time: 16:48)



Example

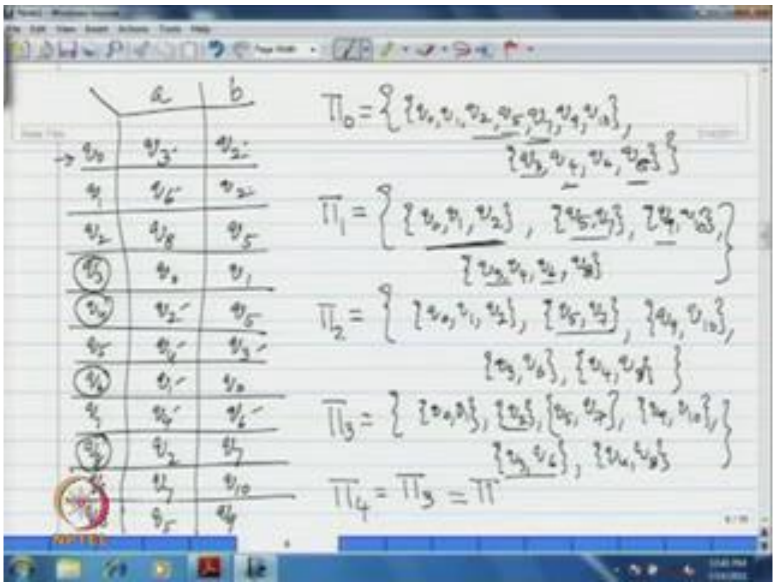
δ	a	b
$\rightarrow q_0$	q_3	q_2
q_1	q_6	q_2
q_2	q_8	q_5
q_3	q_0	q_1
q_4	q_2	q_5
q_5	q_4	q_3
q_6	q_1	q_0
q_7	q_4	q_6
q_8	q_2	q_7
q_9	q_7	q_{10}
q_{10}	q_5	q_9

Formal Languages and Automata Theory

Now, using this result what we understand, the state equivalence can be evaluated by one after another going through calculating one equivalence, two equivalence and so on, at a particular stage if you get pi k is equal to pi k plus 1, then we have obtain the state equivalence partition. And that equivalence is essentially gives you the state equivalence and wherever the more number of states in a particular equivalence class are there, we can remove those things and it take one particular state, and a device a new DFA with less than number of states.

So, whatever the results that I have proved so far let me apply on this particular example and evaluate the state equivalence, so let us consider this following DFA. Here, I give a DFA with ten states in which q 3, q 4, q 6, q 8 are final states with the initial state q naught on the transition are given this way.

(Refer Slide Time: 18:00)



	a	b
$\rightarrow q_0$	q_3	q_2
q_1	q_6	q_2
q_2	q_8	q_5
q_3	q_0	q_1
q_4	q_2	q_5
q_5	q_4	q_3
q_6	q_1	q_0
q_7	q_4	q_6
q_8	q_2	q_7
q_9	q_7	q_{10}
q_{10}	q_5	q_9

$$\Pi_0 = \{\{q_0, q_1, q_2, q_5, q_7, q_9, q_{10}\}, \{q_3, q_4, q_6, q_8\}\}$$

$$\Pi_1 = \{\{q_0, q_1, q_2\}, \{q_5, q_7\}, \{q_9, q_{10}\}, \{q_3, q_4, q_6, q_8\}\}$$

$$\Pi_2 = \{\{q_0, q_1, q_2\}, \{q_5, q_7\}, \{q_9, q_{10}\}, \{q_3, q_6\}, \{q_4, q_8\}\}$$

$$\Pi_3 = \{\{q_0, q_1\}, \{q_2\}, \{q_5, q_7\}, \{q_9, q_{10}\}, \{q_3, q_6\}, \{q_4, q_8\}\}$$

$$\Pi_4 = \Pi_3 = \Pi$$

So, this is the DFA given to us, now first we will calculate this partition, that pi naught zero equivalence partition that means, if you take any two states by apply empty string in both the states, you both are final states or both are non final states, they are zero equivalent. So, that means, in any of the two states for example, in this if I consider q 1 and q 2 if you apply empty string, the resultants states are q 1 and q 2 only, and since both are non final states, they are equivalent.

For example, if you consider q 1 and q 3, if you apply empty string in it, what is happening the resultants states are again q 1 and q 3 only q 1 is an non final state whereas, q 3 is a final state and thus you can understand that, q 1, q 3 are not zero equivalent. Now, you when quickly understand that, what will be the partition pi naught that means, what are the zero equivalence states that we have to write here, all final states will be zero equivalent to each other and similarly all non final states will be zero equivalent to each other.

So, in this partition q naught, q 1, q 2, q 5, q 7, q 9, q 10 these are all non final states, they are all equivalent to each other with respect to zero equivalence. And all final states are here, q 3, q 4, q 6, q 8 they are equivalent to each other, because they are all final states, now pi naught we have calculated zero equivalence partition is here. Now, from zero equivalence partition using the criterion, we can calculate pi 1 that means, one equivalence partition.

What we have to understand here two states, which are already zero equivalent they will be further one equivalent, if you apply any symbol in both the states, if the resultant states are zero equivalent are not that we have to cross check, based on that we have to evaluate this one equivalence partition. Now, you see here we need not verify for q naught, q 3, because this two are not zero equivalent already. So, let us consider a particular equivalence class in which, what are all the states further they will continue to be one equivalence, that is what we have to verify.

Now, let us consider q naught and q 1, since they are zero equivalent and you see this q naught, q 1, if you apply a the resultant states are q 3 and q 6, you can see here q 3 and q 6. In q 3 is a final state, I mean q 3 and q 6 are in the equivalence class, then you check for the symbol b, for the symbol b the resultants states are q 2 and q 2 and thus I can say that q naught, q 1 are further one equivalent, so they will be in the same equivalence class.

Now, I will verify q 1 and q 2 by transits you can quickly understand whether q naught is, if they are not equivalent, if q 1 and q 2 are not equivalent then q 2 cannot be equivalent to q naught also, because q naught and q 1 are one equivalent already. So, q 1 and q 2 you consider and you see when you apply a, that is q 6 and q 8, q 6, q 8 are in same equivalence class and q 2, q 5 are in the same equivalence class here, and thus q 2 is also will be in the same equivalence class with q 1 and q naught.

Now, let us take q 2, q 5, if you consider q 2, q 5 when apply a the resultants states q 8, q 4 and you see q 8, q 4 in the same equivalence class here and when you apply b that is q 3 and q 5, but you see q 5 is in this equivalence class whereas, q 3 is in this equivalence class. So, at this symbol the resultant states since they are not zero equivalent, we understand that q 5 is not one equivalent to q 2 and hence q 5 has to be put in separate equivalence class.

So, let me start a new equivalence class for this purpose, so a q 5 are you understand that q 5 cannot be equivalent to q 1 and q naught also, because q 2 is equivalent to q 1 and q naught, now let us start a new equivalence class for this. Now, let us consider q 5, q 7 or you can check whether q 7 is equivalent is q 2 and if it so then it will you can put q 7 in this equivalence class, if it is not then you will cross check with q 5. And if they are equivalent, then you put in that equivalence class, even otherwise you can start a new equivalence class for a q 7.

Let us take q 7 and check the equivalence with q 2, so q 2, q 7 the resultant states are q 8, q 4 when you apply a and when you apply b that is q 5, q 6. You see here, when I am cross checking with q 2 and q 7 at b q 5, q 6 have come as a resultant states, and you see q 5 is in first equivalence class and q 6 is in the second equivalence class, thus q 2 and q 7 are not equivalent.

So, q 7 I cannot put in this equivalence class, first equivalence class, now let me cross check whether q 7 is equivalent to q 5, I mean one equivalent to two q 5. So, q 5, q 7 and I am cross checking by supplying the symbol a, resultant states are q 4 they will be in the same equivalence class. And here then you apply b that is q 3, q 6 and you see q 3 and q 6 are also in the same equivalence class and hence q 7 is one equivalent to q 5.

So, I will be putting this q 7 in the equivalence class of q 5, now let us take q 9 with q 7, q 9, q 7 and I am considering and I apply a I am getting q 4, q 7. And you see q 4 is in this equivalence class, whereas q 7 is in this equivalence class and hence, q 9, q 7 are not one equivalent. Now, let me cross check whether q 9 will go in this equivalence class, the equivalence class of q naught, q 1, q 2 let me cross check, so let me cross check with one of them.

Take q 9, q 2 and apply a the resultant states are q 7, q 8 and you understand that q 7 is in this equivalence class and q 8 is in the equivalence class, so q 9 cannot go along with q 2 also, thus I have to start for q 9 a new equivalence class, so let me start that q 9. Now, in this equivalence class I have q 10, now whether q 10 will fall in the first equivalence class or in the second one or third one let us cross check.

So, let me cross check this q 10, q 9 whether they are equivalent, in q 9 and q 10 if you apply a the resultant states are q 5, q 7 and you see they are in the same equivalence class. And if you apply b that is q 9, q 10, you see they are in the same equivalence class and hence, I am getting that q 10 is one equivalent to q 9, so we can put in this equivalence class.

Now, I can close this classes, because nothing further you will add, because what are all the states in this second equivalence class of pi naught, none of them will go along in any of this classes. So, I will close these things, now let us cross check whether q 3 and q 4 are further one equivalent, q 3, q 4 when I am cross checking the resultant states are q naught, q 2 and you see q naught and q 2 are zero equivalent. And when I apply b q 1, q 5 are coming and you see q 1 and q 5 are zero equivalent and thus q 3, q 4 they should be in same equivalence class.

Now, let me cross check q 4, q 6, if you apply a we are getting q 2, q 1, you see here q 2, q 1 we are getting and q 2, q 1 are zero equivalent. And when you apply b I am getting q 5, q naught they are also zero equivalent and thus q 6 will be the equivalence class of q 4. Now, let us see what about q 8, let us cross check with q 8 with q 6, in q 8 q 6 if you apply a we are getting q 1 q 2, they are zero equivalent I mean you apply b q naught q 7, so they are also zero equivalent.

And thus I can see that q 8 is also in the same equivalence class, thus this is the one equivalence partition pi 1, we have calculated one equivalence partition pi 1, now you can continue this fashion to calculate pi 2. Without much explanation very quickly I will see that you can q naught, q 1 you we cross check and you understand that q naught, q 1 will be equal and you just cross check that.

And q 1, q 2 you cross check and you understand that q 2 will also be equivalent to this and that is I am putting here, and q 5, q 7 you cross check you understand the resultant states are one equivalent and thus i can put again they are continuing into this same equivalence class. If you want a verification here, q 5, q 7 you take the resultant states are q 4 when you apply a, when apply b q 3, q 6 and you see q 3, q 6 are in the same equivalence class of one equivalence that in pi 1, here q 3, q 6.

And thus I am putting q 5, q 7 continuing to be in a same equivalence class, similarly if you evaluate you will understand that q 9, q 10 will continue to be equivalent. Whereas, here what you will understand is q 3, q 6 are equivalent whereas, this q 4 is not equivalent to q 3 and it will form a new equivalence class and this is the equivalence class that you will get, and this is the corresponding partition of two equivalence.

So, pi 2 once again you verify as per the process and now let us calculate pi 3 and in pi 3, if you once again follow this q naught, q 1 you cross check and you understand that they are equivalent. Whereas if you cross check q 1 with q 2 you understand that, this two will not be three equivalent and q 2 gets separated, so it will form a another equivalence class and q 5, q 7, if you cross check there will be further three equivalent, so q 5, q 7 will continue to be equivalent in three equivalence partition.

And this q 9, q 10 if you cross check they will further be equivalent and now you cross check q 3, q 6 you will understand that they are further three equivalent, q 3 q 6. And similarly, q 4, q 8 if you cross check, the resultant states are two equivalent and therefore, these two states are three equivalent, so if you calculate you will get this is the three equivalence partition.

Now, since there is difference between pi 2 and pi 3, because in pi 2 q naught, q 1, q 2 are equivalent to each other, here q 1 and q 2 are not equivalent. So, thus there is difference between pi 2 and pi 3, there is difference from pi naught to pi 1, pi 1 to pi 2 and pi 2 to pi 3. So, now, we have to further continue, let us now calculate pi 4 use the same process that means, what are all the states three equivalent, you just cross check using that criteria, whether they are four equivalent.

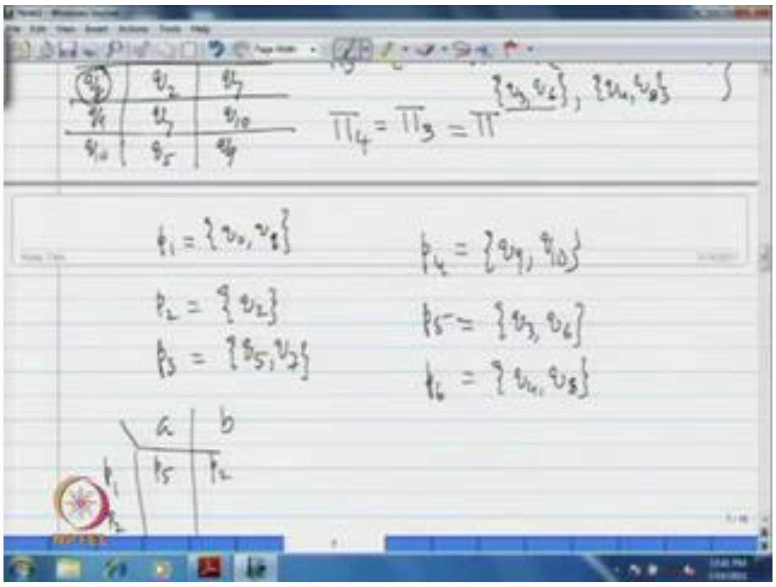
That means, q naught, q 1 you take and see by applying a, applying b whether they are further equivalent or not for example, you have q 3, q 6 in q naught, q 1 if you put q 3, q 6 are three equivalent and if you apply b that is q 2. So, that is anyway the same single state thus q naught and q 1 further continue to be four equivalent and of course, q 2 is a single term set.

So, that will continue to be single equivalence class q 5, q 7 when you are cross checking, you will understand q 5, q 7 the resultant states are when you apply a q 4 and when you apply b q 3, q 6 and you understand q 3, q 6 are three equivalent and this q 5, q 7 continue to be four equivalent. Similarly, you cross check for q 9 q 10, q 3 q 6, q 4 q 8 and understand that, these are all further four equivalent.

And that means, here pi 4 is actually equal to pi 3 and thus we have got a situation that pi k is equal to pi k plus 1, here k equal to three, so after this minutes, so one, two three steps we understand. And the four equivalence partition is equal to three equivalence partition that means, from the previous theorem if for some k, pi k is equal to pi k plus 1, then they are that is the equivalence partition, state equivalence partition.

So, here we understand that we have calculated the state equivalence partition pi that is what is pi 3, now what do you do from this equivalence classes, you pick up 1 of the states. For example, in this equivalence class you can q naught, in this anyway you take q 2, in this q 5, in this say for example, q 9, q 3, q 4 you take; and the respect you transition you maintain, thus you will be able to write a new DFA. Now, of course, whatever the new DFA that I am writing by removing the equivalent states, whether that is equivalent to the original DFA or not that we have to cross check, so this is how we are reducing the states. Instead of removing some other states you can re-label them for example, q naught, q 1, this q naught, q 1 you mean call it as say p 1.

(Refer Slide Time: 34:09)



q_8	q_2	q_7
q_9	q_7	q_{10}
q_{10}	q_5	q_9

$\{q_3, q_6\}, \{q_4, q_8\}$ }

$\Pi_4 = \Pi_3 = \Pi$

$p_1 = \{q_0, q_1\}$

$p_2 = \{q_2\}$

$p_3 = \{q_5, q_7\}$

$p_4 = \{q_9, q_{10}\}$

$p_5 = \{q_3, q_6\}$

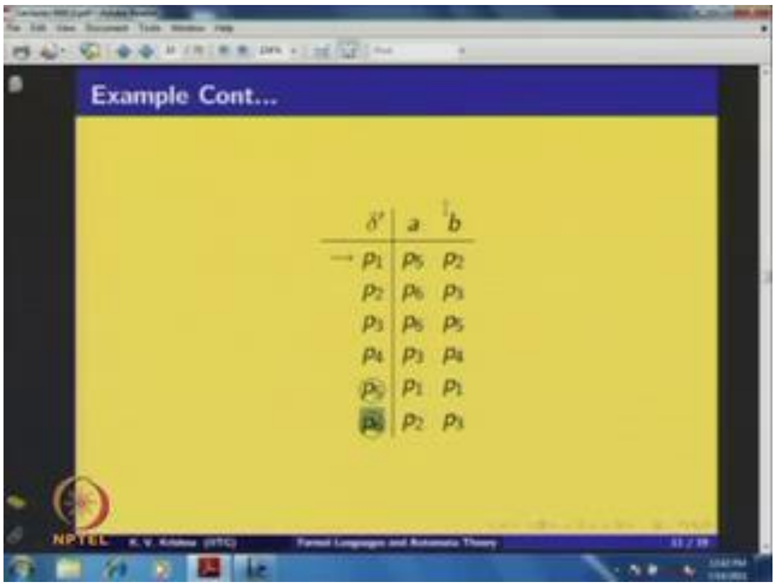
$p_6 = \{q_4, q_8\}$

	a	b
p_1	p_5	p_2
p_2		

Let me, so this state q naught, q 1 I call it as say p 1 and p 2, I will take the single term q 2, p 3 I take this q 5, q 7, p 4 I take q 9, q 10 and p f 5, I will take this q 3, q 6, p 6 this is relabeling of the resultant equivalence classes q 4, q 8. So, I have got six equivalence classes, I am calling them p 1, p 2, p 3, p 4, p 5, p 6, now what do you do the original DFA corresponding to those transition take a, b take p 1, p 2, p 3, p 4, p 5, p 6 a states.

And now for transitions corresponding to p 1, in this you pick up anyone of the states, pick up either q naught or q 1 and look at the transitions, for q naught we have q 3 and if you apply b you get q 2 and in this equivalence classes, q there is falling in this equivalence class and q 2 is falling in this equivalence class, so those equivalence classes levels you can write. So, q 3, q 6 this is p 5 and this is p 2, so this is how the resultant transitions we write and similarly, if you continue here it is p 6 this is p 3. And for p 3 also you write p 4, p 5 and p 6, if you write this table by taking the transition, because for p 2, this is q 2 corresponded to q 2 what are all the transition the resultant states in which equivalence classes that they are falling, you write similarly for p 3 if you write you get p 6, p 5.

(Refer Slide Time: 36:30)

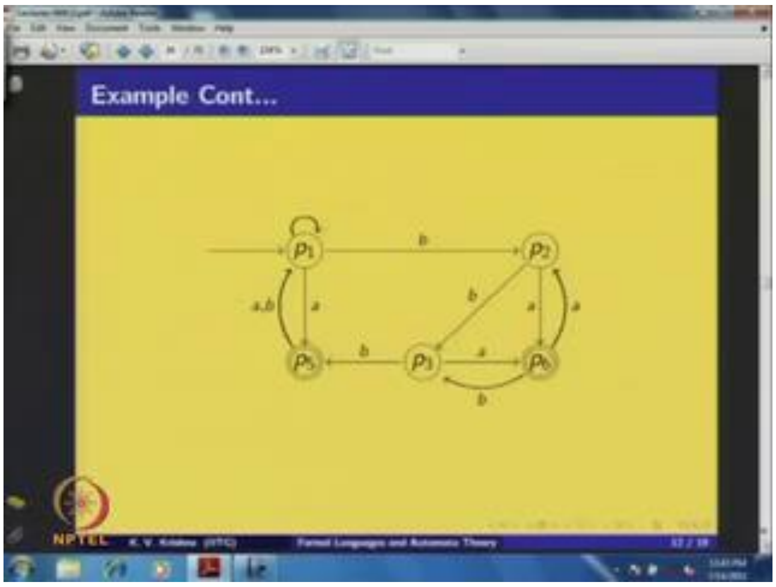


Example Cont...

δ'	a	b
$\rightarrow p_1$	p_5	p_2
p_2	p_6	p_3
p_3	p_6	p_5
p_4	p_3	p_4
p_5	p_1	p_1
p_6	p_2	p_3

And if you continue this you get this is the transition table, the transition table and since q naught is the initial state and q naught is falling in the equivalence class. p 1, we discrete p 1 as initial state and final states are falling in p 5 and p 6 and thus we will make this p 5 and p 6 as final states in the new DFA. So, thus we will construct a DFA by reducing the states wherever there are some equivalent states, now the question is what are the DFA that I have formally if you draw the state diagram of this, this is the resultant DFA.

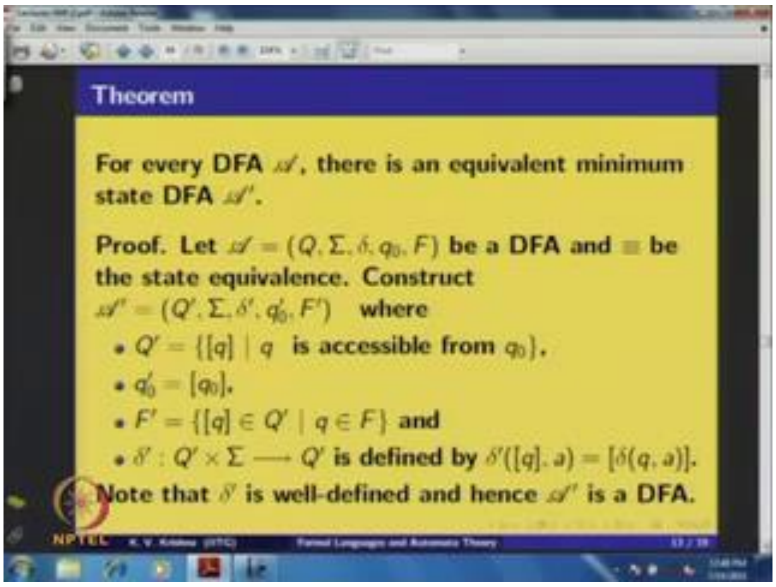
(Refer Slide Time: 37:13)



Example Cont...

Now, the question is what are the DFA that we have constructed in this manner, whether this is minimal DFA, minimal means further can be reduced the states here as a question here. Now, when I am considering this equivalence classes, now let us look at p 1, so these are the classes that we have consider, now let us see relate to that following theorem.

(Refer Slide Time: 37:53)



Theorem

For every DFA $\mathscr{A}$, there is an equivalent minimum state DFA $\mathscr{A}'$.

Proof. Let $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA and $\equiv$ be the state equivalence. Construct $\mathscr{A}' = (Q', \Sigma, \delta', q_0', F')$ where

- $Q' = \{[q] \mid q$ is accessible from $q_0\}$.
- $q_0' = [q_0]$.
- $F' = \{[q] \in Q' \mid q \in F\}$ and
- $\delta' : Q' \times \Sigma \longrightarrow Q'$ is defined by $\delta'([q], a) = [\delta(q, a)]$.

Note that δ' is well-defined and hence $\mathscr{A}'$ is a DFA.

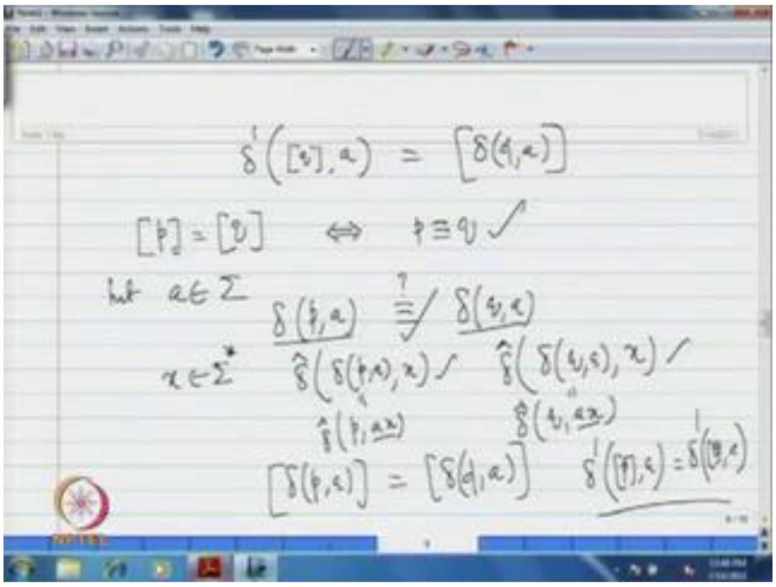
For every DFA A, there is an equivalent minimum state DFA A dash, how do we do that what are the algorithm approach that we have adopted just now, that means, by calculating the state equivalence through, one equivalence, two equivalence, three equivalence and so on, we are getting straight equivalence from the state equivalence we have formatted the DFA. So, we will follow that method and we first constructed DFA and we observe that what are that we have constructed that is in fact, is a minimum state DFA, corresponding to that language.

Of course, an equivalent means two automata are equivalent, if the language expert is both the automata are same. So, let us consider A, Q, sigma, delta, q naught, F a DFA and take the state equivalence on this and calculate the partition. Now, let us construct a dash with Q dash, sigma, delta dash, q naught dash, F dash where this state set is the equivalence classes with respect to this equivalence relation.

And moreover what are all those states which are accessible from q naught, if a particular state is not accessible, even if it is the equivalence class corresponding to the particular one, since it will not contribute to the language, you can simply ignore those equivalence classes. So, consider those equivalence classes, if q is accessible from q naught in the original DFA, now what is a initial state, the equivalence class containing the initial state of the original DFA, as we have constructed earlier in that example.

Now, what are the final states consider those equivalence classes which are having final states, now what is the transition, the transitions delta dash is from q dash cross sigma to q dash which is defined by. In a equivalence class if you apply a, in the original DFA you apply a in that particular and see what are the resultant equivalence class and assign to that. Now, the way that I am defining here whether this is well defined that is what we have to cross check.

(Refer Slide Time: 40:40)



$$\delta'([q], a) = [\delta(q, a)]$$

$$[p] = [q] \Leftrightarrow p \equiv q \checkmark$$

let $a \in \Sigma$ $\quad \underline{\delta(p, a)} \overset{?}{\equiv} \checkmark \underline{\delta(q, a)}$

$x \in \Sigma^*$ $\quad \hat{\delta}(\delta(p, a), x) \checkmark \quad \hat{\delta}(\delta(q, a), x) \checkmark$

$$\hat{\delta}(p, ax) \qquad \hat{\delta}(q, ax)$$

$$[\delta(p, a)] = [\delta(q, a)] \qquad \delta'([p], a) = \delta'([q], a)$$

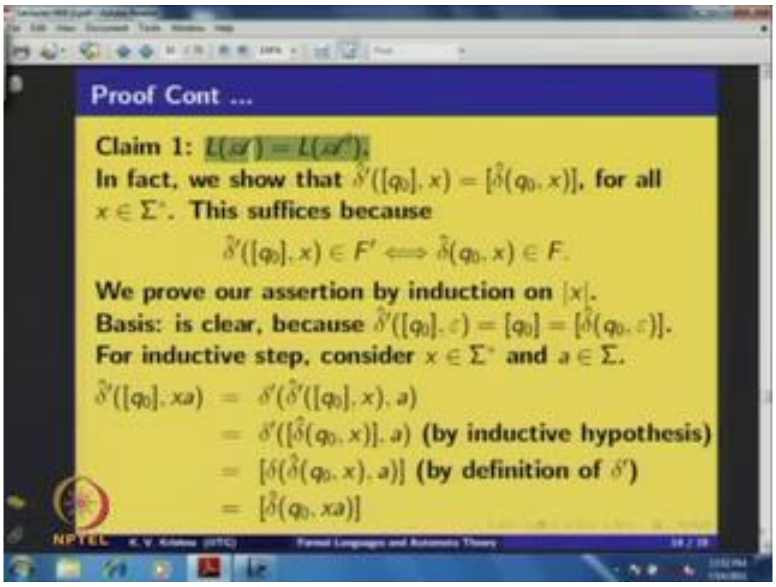
So, what we are defining here is delta dash of bracket q at a is defined as in the original DFA, in the state q you apply a and consider the equivalence class containing that. Now, to check whether this is delta dash is well defined, consider two equivalence classes say bracket p is equal to bracket q. That means, p and q are equivalent to each other that means, if you put any string in both p and q both are final states or non final states, that is the meaning here.

Now, what I have to understand delta dash of p comma a and delta dash of q comma a they are same, so take fix let a belongs to sigma, now you look at delta of p comma a and delta of q comma a, whether they are equivalent or not. That means, if you chose any string x and sigma star what we have to cross check that this delta cap of delta of p comma a at x, and a delta cap of delta of q comma a at x, the resultant states both are final states or non final states that is what we have to understand.

Now, this is nothing else, but delta of p comma a x and this is delta cap of q comma a x, now since p and q are equivalent to each other by applying any string both in both p and q, both are as a final states or non final states. And thus you see since this is equal to this, this two states by applying x either final states or non final states and hence, you understand that delta of p comma a and delta of q comma a are equivalent to each other.

And hence, the corresponding equivalence classes are same delta of p comma a, the equivalence class is equal to, the equivalence class containing delta of q comma a and hence, delta dash of bracket p at a is same as delta dash of bracket q at a. And hence, delta dash is well defined, so this map is well defined and hence, you understand that this is a DFA whatever we have constructed a dash that is a DFA.

(Refer Slide Time: 43:30)



Proof Cont ...

Claim 1: $L(\mathcal{A}) = L(\mathcal{A}')$.
In fact, we show that $\hat{\delta}'([q_0], x) = [\hat{\delta}(q_0, x)]$, for all $x \in \Sigma^*$. This suffices because

$$\hat{\delta}'([q_0], x) \in F' \iff \hat{\delta}(q_0, x) \in F.$$

We prove our assertion by induction on $|x|$.
Basis: is clear, because $\hat{\delta}'([q_0], \varepsilon) = [q_0] = [\hat{\delta}(q_0, \varepsilon)]$.
For inductive step, consider $x \in \Sigma^*$ and $a \in \Sigma$.

$$\begin{aligned} \hat{\delta}'([q_0], xa) &= \delta'(\hat{\delta}'([q_0], x), a) \\ &= \delta'([\hat{\delta}(q_0, x)], a) \text{ (by inductive hypothesis)} \\ &= [\delta(\hat{\delta}(q_0, x), a)] \text{ (by definition of } \delta') \\ &= [\hat{\delta}(q_0, xa)] \end{aligned}$$

Now, the newly construct a DFA A dash is equivalent to the original DFA that means, we have to cross check L A is equal to L A dash both the languages are same. In fact, we show that this delta dash of bracket q naught x is equal to the equivalence class containing this state, delta dash of in the original automaton in the initial state if you apply x, what are the state that you are getting we consider the equivalence class.

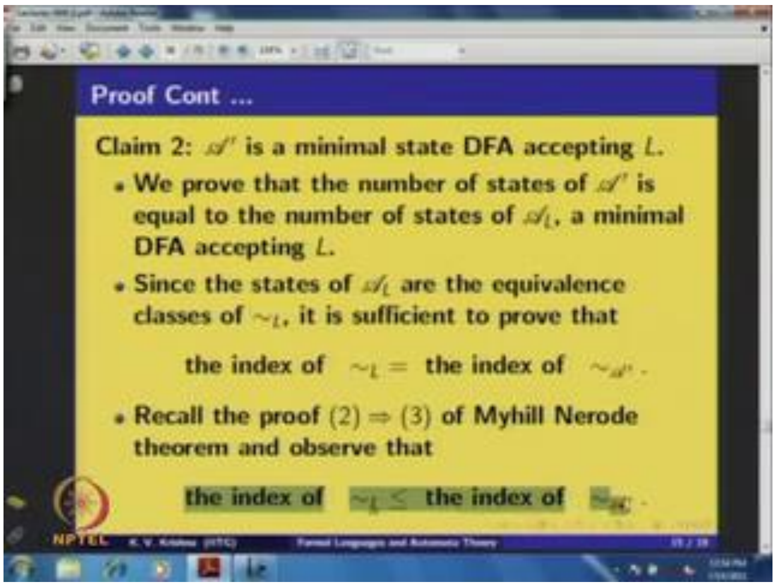
So, in fact, we show this thing, if we show this then it is sufficient, because if this resultant state this is an F dash that means, in the final state, that mean this delta dash of q naught comma x should be in F, that is how we have consider F dash and of course, conversely. Because of this any string, if you apply in initial state, if it goes to final state in the original DFA that means, it is in the language accepted by a and thus if you apply this string in the initial state of the DFA A dash, then you will be in the final state.

So, because of this reason you can understand that, if you we show that this is happening then the language x by a and language x A dash are same, because those strings which are going to final states in the original DFA in A dash also, they will go to final states they will, they will take the automaton to final states and vice versa. We prove this assumption by index on the length of x, but the base is for this induction is very clear, because if you consider this, this is what is when you apply empty string, this is bracket q naught in a DFA.

And bracket q naught is nothing else, but delta cap of q naught at epsilon by applying a epsilon in the original DFA and cancel the equivalence class, so the base is for the induction is very clear. Now, for in the two step you consider a string x and sigma star and a symbol a in sigma and consider this delta cap at q naught at x a, and you see this is equal to, this is the expansion when you apply any string and a by inductive hypothesis this guy is the equivalence class containing the state. Delta dash at bracket q naught at x is the equivalence class containing delta of q naught at x, so this is by inductive hypothesis.

And now if you apply a here, from definition of delta dash this is the equivalence class containing this and that means, what you have the equivalence class containing the state, when you apply x a in q naught of the original automaton. So, as desired we have got delta dash of q naught at x a, the equivalence class containing q naught at x a is equal to the equivalence class of delta of q naught at x a. And thus from induction we can conclude that, we have got this thing and hence, the language absolute by a is same as language absolute by A dash.

(Refer Slide Time: 46:52)



Proof Cont ...

Claim 2: $\mathscr{A}'$ is a minimal state DFA accepting L.

- We prove that the number of states of $\mathscr{A}'$ is equal to the number of states of $\mathscr{A}_L$, a minimal DFA accepting L.
- Since the states of $\mathscr{A}_L$ are the equivalence classes of $\sim_L$, it is sufficient to prove that

  the index of $\sim_L =$ the index of $\sim_{\mathscr{A}'}$.
- Recall the proof (2) $\Rightarrow$ (3) of Myhill Nerode theorem and observe that

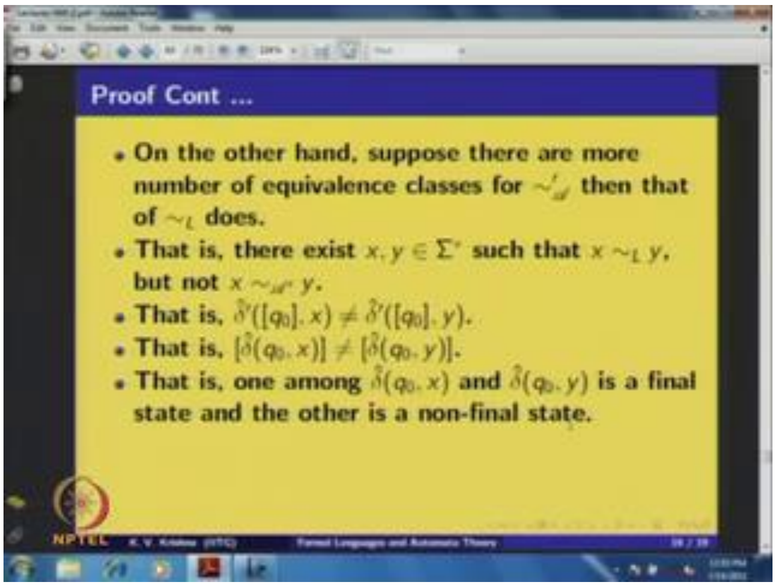
  the index of $\sim_L \leq$ the index of $\sim_{\mathscr{A}'}$.

So, what are the new automaton that we have constructed that is equivalent to the original DFA, now the second point we have to observe that this is having the minimum number of states. To prove that A dash is a minimal state DFA accepting L, what we do we prove the number of states of A dash is equivalent to the number of states of A L, you recall A L is the minimal DFA accepting L, is the Myhill-Nerode theorem.

Whatever the this the automaton A L, we have constructed in the last step of the proof, there we have called it as A L and you remember that is the minimal DFA accepting L. Now, what we prove here A dash is having the exactly the same number of states as A L, and since we know A L is minimal DFA accepting L we understand, this A dash is also minimal state DFA. Since state of A L are the equivalence classes of this tilde L, because this is equivalence relation we have associated to the language L,, now it is sufficient to prove that index of this tilde L is same as the index of tilde A.

That means, there equivalence classes of tilde L and equivalence classes of tilde A dash they are same, that is not we will prove, and now recall the proof of, proof 2 implies 3 of Myhill-Nerode theorem. In which what we have proved the index of tilde L is always less than equal to index of tilde A dash, it is not only for A dash for any automaton which is accepting L, so we have this already.

(Refer Slide Time: 48:45)



Proof Cont ...

- On the other hand, suppose there are more number of equivalence classes for $\sim'_{\mathscr{A}}$ then that of $\sim_L$ does.
- That is, there exist $x, y \in \Sigma^*$ such that $x \sim_L y$, but not $x \sim_{\mathscr{A}'} y$.
- That is, $\hat{\delta}'([q_0], x) \neq \hat{\delta}'([q_0], y)$.
- That is, $[\hat{\delta}(q_0, x)] \neq [\hat{\delta}(q_0, y)]$.
- That is, one among $\hat{\delta}(q_0, x)$ and $\hat{\delta}(q_0, y)$ is a final state and the other is a non-final state.

On other hand, if we assume the index of this is more than that of tilde L, because we want to observe this two are equal, on other hand if you assume there are more number of equivalence classes of this tilde A dash, this prime is wrongly typed this you have to have A dash here. Then spelling mistake, then that of tilde a tilde L dash that is, so if you have equivalence classes with respect to this equivalence relation, what is the meaning of this you can find two strings x, y in sigma star such that, there equivalent with respect to tilde L.

But, they are not equivalent with respect to this tilde A dash, that is because tilde A dash in a DFA the equivalence relation that, if you apply in the initial state, if both the resultant state are same by applying x and y. Then, we say they are equivalent, here they are not equivalent that means, if you apply in the initial state that is bracket q naught here, if you apply x what are the state that you are getting it is different from when you apply y in the initial state bracket q naught. That means, by definition just now we have we have understood that this value is equal to this, the equivalence class containing this state. So, this two are not equal, that is one among these two equivalence classes are not same means, the state in this by the one among the states, this is a final state and other is non final state.