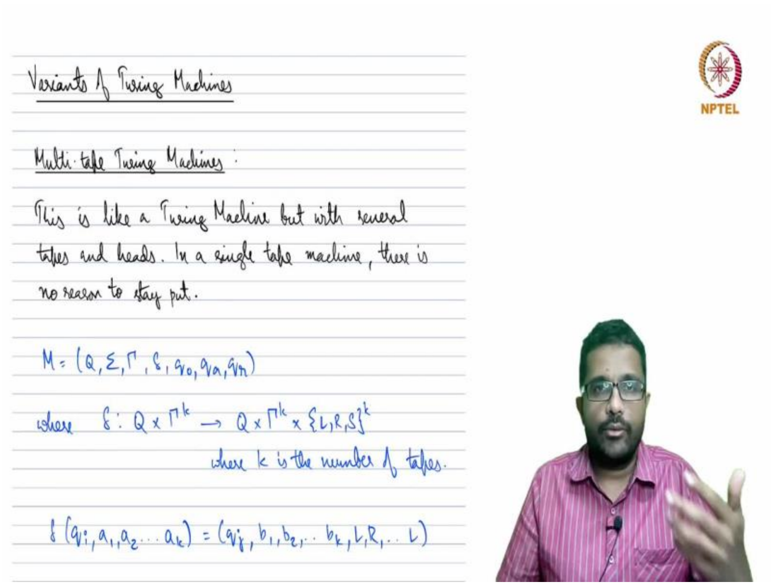
Theory of Computation

Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Multitape Turing Machine

(Refer Slide Time: 0:17)



Variants of Turing Machines

Multi-tape Turing Machines:

This is like a Turing Machine but with several tapes and heads. In a single tape machine, there is no reason to stay put.

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

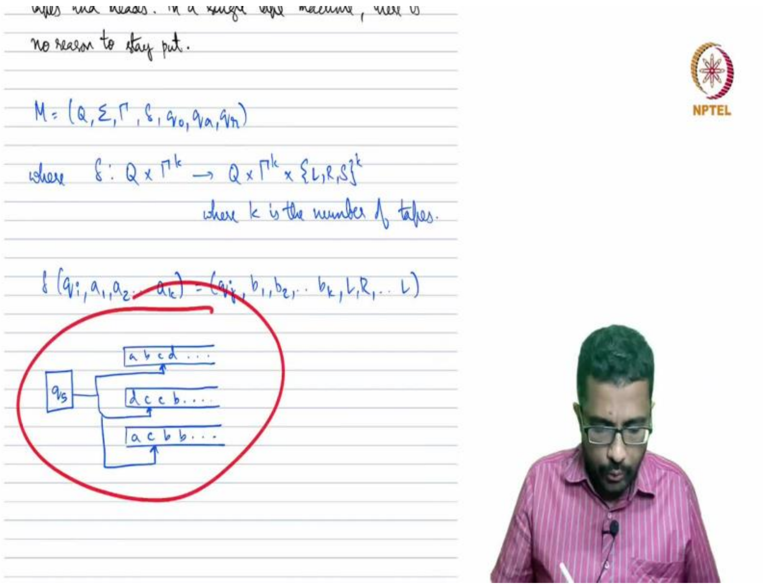
where $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

where k is the number of tapes.

$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$

Hello and welcome to lecture 29 of the course Theory of Computation. We have been seeing Turing machines. So far, we have seen the basic model of the Turing machine which is a deterministic Turing machine with a single tape. Now, we will see other variants of Turing machines. In this lecture, we will see Multi-tape Turing machines. So, what are Multi-tape Turing machines? They are just what you would imagine the name suggests. Instead of a single tape, we will have multiple tapes.

(Refer Slide Time: 0:50)



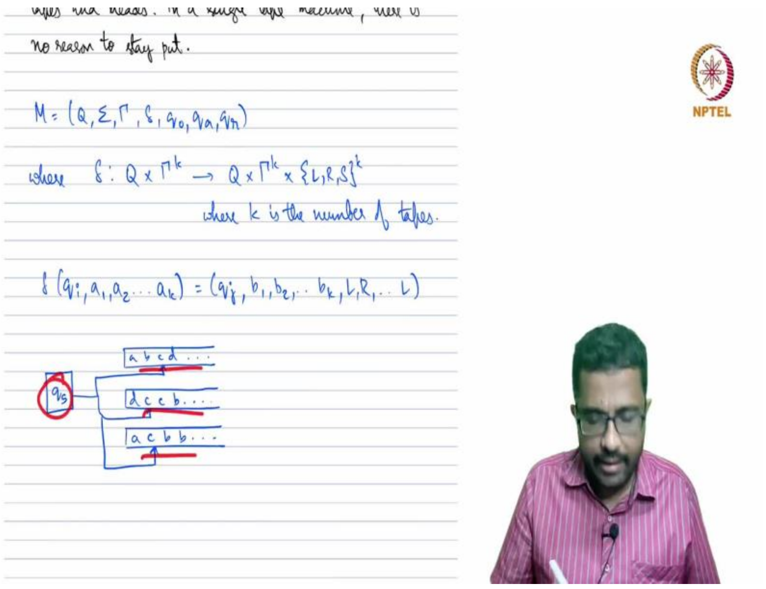
no reason to stay put.

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

where $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$

where k is the number of tapes.

$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$



no reason to stay put.

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

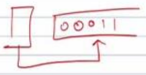
where $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$

where k is the number of tapes.

$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$

Variants of Turing Machines

Multi-tape Turing Machines :



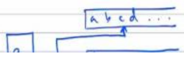
This is like a Turing Machine but with several tapes and heads. In a single tape machine, there is no reason to stay put.

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

where $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$

where k is the number of tapes.

$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$



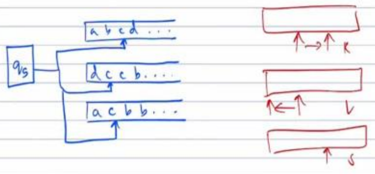
tapes and heads. In a single tape machine, there is no reason to stay put.

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

where $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$

where k is the number of tapes.

$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$



This is like a Turing Machine but with several tapes and heads. In a single tape machine, there is no reason to stay put.

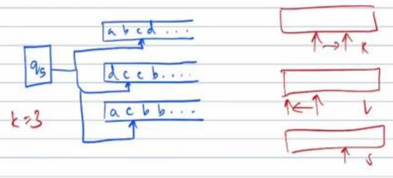
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

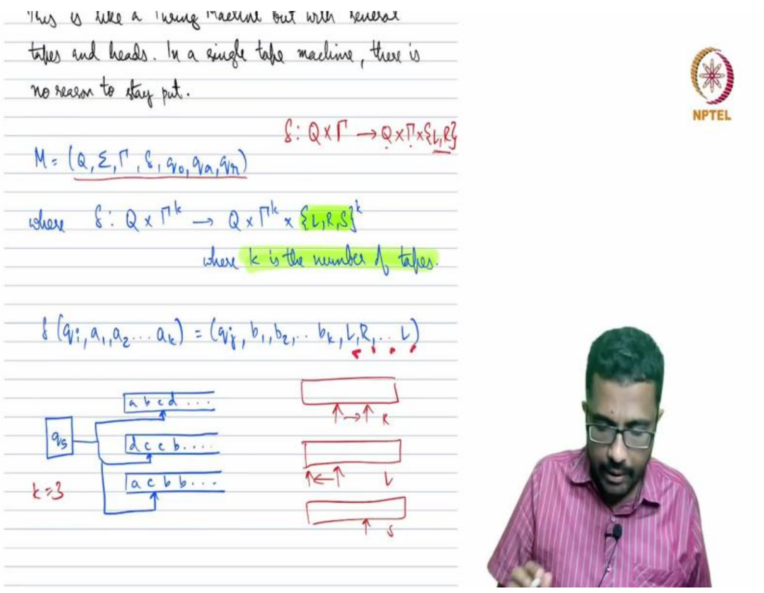
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

where $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$

where k is the number of tapes.

$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$





This is like a Turing Machine but with several tapes and heads. In a single tape machine, there is no reason to stay put.

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

where $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

where k is the number of tapes.

$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$

So, like what we have here, you have one state control, and you could have multiple tapes and with each tape containing different possible contents. And the heads of each of these tapes do not have to move in the same manner. In each step of computation, the tape one the head could move right, the tape 2 head could move left and tape 3 the head could… based on whatever the rules are, the rules allow each head to move in a different direction.

So, it is like a Turing machine. But instead of a single tape, it can have multiple tapes. And for each tape, there will be a head. So, one thing that I would like to note here is that in the basic model in the single tape Turing machine, we had a state control and we had a tape and a head. And the only options that we allowed were left and right for the movements of the head. In this case, we are going to allow one more option, which is the stay option. So, the tape head is also allowed to stay.

This is because in the case of a single tape, there is nothing that gets accomplished by allowing the head to stay. Because if you stay at the same spot, you must stay there for some time, but eventually you will have to move. So, if you stay there for 3 moves, and finally you write a 1 in that location and you move, so you might as well have written a 1 in the first instance itself and then made the move. There was nothing gained by staying in the same place for a while because it is not really furthering or progressing the computation.

Whereas in the case of multi tape Turing machine, the stay option becomes meaningful. Because you could have, maybe a move in such a way that in the first tape, the head has to move to the right side. And in the second tape, the head has to move the left side. And in the third tape, we just want the head to stay wherever it was. So, this is a right, this is a left and this is a stay and we want this option, because we do not necessarily want each head to move in all the moves. We may want a subset of the heads to move.

So that is why a stay option becomes meaningful in the case of a Multi-tape Turing machine. Because we want to allow the possibility of a head remaining in the same place because some other head may be moving. In the case of single tape Turing machine, if we allow that possibility, it means that there is no progress in the computing because there is only one head, and that head remains in staying put or remaining in the same place means that the computation can progress only when that head moves.

So that is why in the case of a single tape Turing machine, we did not allow the stay option. Whereas here it makes sense to include the stay option. So let us just see the definition. I will not get to the detailed definition, but I will just say the brief definition or only highlight the differences. So, we have $M = (Q, \Sigma, \Gamma, \delta, \mathrm{q}_0, \mathrm{q}_\mathrm{a}, \mathrm{q}_\mathrm{r})$, the Turing machine being the 7 tuple. Q_0 being the start state, q accept and q reject being $\mathrm{q}_\mathrm{a}, \mathrm{q}_\mathrm{r}$. And so, in the case of a single tape Turing machine, we had the transition function, $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.

So, you are at a certain state. You read the tape input, tape content at that position where the head is pointing. And then you have to decide which state to go to, what to write on the tape. So which state to go to is this Q, what to write is this Γ, and then L or R, whether the tape head should move to the left or the right. Here, it is very similar. But notice that here, instead of one tape, we are allowing k number of tapes. So, there could be multiple tapes.

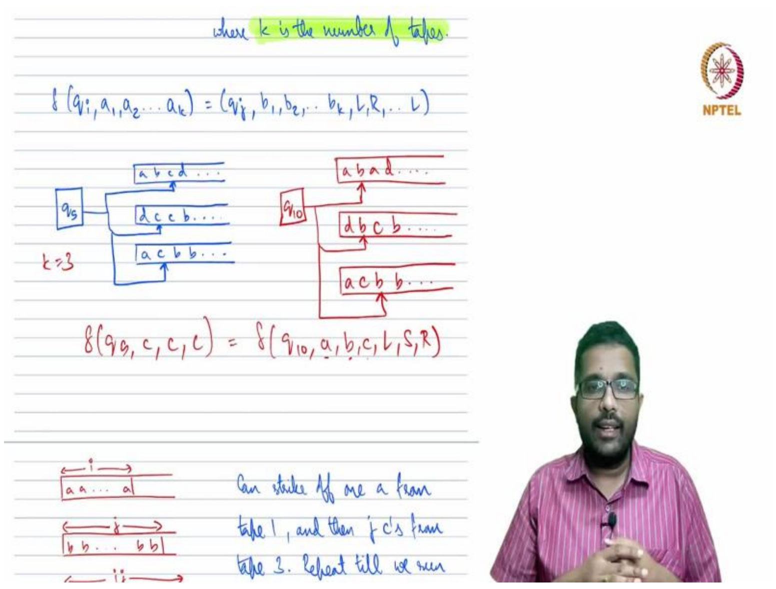
So, in the figure here, k is 3. There are 3 tapes here, so here k is 3. So, the moves will be decided based on the current state and what all the tape heads read. So here, it would be decided based on what all the 3 tape heads read. So that is why we have Γ^k here.

$$\delta: Q \times \Gamma^\mathrm{k} \rightarrow Q \times \Gamma^\mathrm{k} \times \{L, R, S\}^k$$

So, based on all the k tapes, it is what they read. Then we have to decide which state to move to? What to write on all the k tapes and L, R, S. So here, I have already explained the meaning or why we need the stay option. For each of the tape heads, we have to say whether they move left, right or stay. So that is the main difference. So first of all, this Γ and this Γ and this L are they become k fold. And the second difference is that in addition to L and R, we allow a stay option S. These are the 2 differences.

So, the δ is of this form, $\delta(q_i, a_1, a_2 \dots, a_k) = \left(q_j, b_1, b_2, \dots, b_k, L, R, \dots L\right)$. Meaning these are the tape contents that you read, $a_1, a_2 \dots, a_k$ and q_i is the current state. Then q_j is the next state, then use say $b_1, b_2, \dots, b_k$ as the symbols that you write and L, R etc. be the movements that you make. So maybe just to see an example, I will just erase this and we will see an example.

(Refer Slide Time: 07:05)

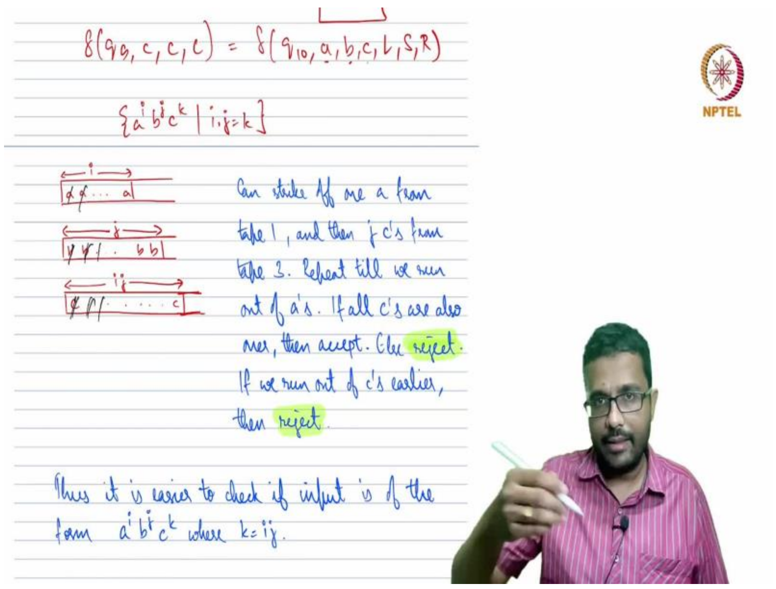


where k is the number of tapes.

$$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$$

$$\delta(q_9, c, c, c) = \delta(q_{10}, \underset{-}{a}, \underset{\cdot}{b}, \underset{\cdot}{c}, L, S, R)$$

Can strike off one a from tape 1, and then j c's from tape 3. Repeat till we run



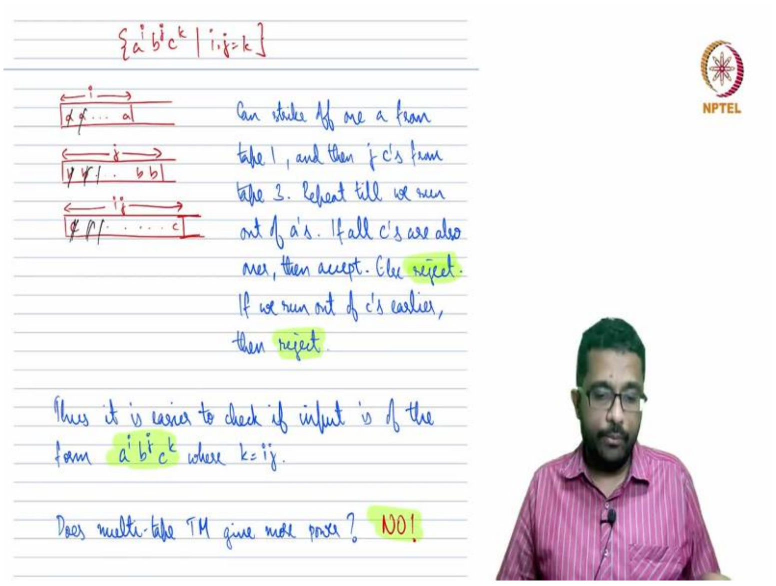
$$\delta(q_9, c, c, c) = \delta(q_{10}, \underset{-}{a}, \underset{\cdot}{b}, \underset{\cdot}{c}, L, S, R)$$

$$\{a^i b^j c^k \mid i.j = k\}$$

i
$\cancel{a}\cancel{a}$ . . . a
j
$\cancel{b}\cancel{b}$. b b
ij
$\cancel{c}\cancel{c}$ c
NPTEL

Can strike off one a from tape 1, and then j c's from tape 3. Repeat till we run out of a's. If all c's are also over, then accept. Else reject. If we run out of c's earlier, then reject.

Thus it is easier to check if input is of the form $a^i b^j c^k$ where $k = ij$.


$\{a^i b^j c^k \mid i.j = k\}$

Can strike off one a from tape 1, and then j c's from tape 3. Repeat till we run out of a's. If all c's are also over, then accept. Else reject. If we run out of c's earlier, then reject.

Thus it is easier to check if input is of the form $a^i b^j c^k$ where $k = ij$.

Does multi-tape TM give more power? NO!

So let us say δ, so here it is state is q5, c is the first tape, c the second symbol, in fact c is the third symbol coincidently and let us say this was $\delta(q_{10}, a, b, c, L, S, R)$. So, what I am saying is if this is the situation, then you move to q_{10}. The first tape you write a. So, a b, this head was pointing at c so now this becomes a, the second tape you write b. It was pointing at c so it becomes d b c b. Wherever the head was not pointing those symbols are not changed.

And the third tape, it was pointing at c and that remains c so you have a c b b. Wherever the head was not pointing, that does not change. And further, the first tape moves one step to the left. The second head stays and the third head moves one step to the right. So, this is what one move of a 3 tape Turing machine would look like.

So, sometimes having multiple tapes affords us some conveniences. In the sense that you may have to read something which is on the one end of the tape. But now this allows some kind of convenience in terms of the memory organization. So, by having multiple heads, it allows us to organize things more efficiently. And this can make the algorithm simpler. So let us see one simple example.

$$\{a^i b^j c^k | i.j = k\}$$

So, remember, we had this language in the previous lecture. This is something similar. i multiplied by j equal to k. So now, if we have to somehow accomplish that. So how it happens is usually that the input is in the first tape of the entire input. And then we use the other tapes as some kind of working tapes. So, suppose this was the entire input, $a^i b^j c^k$. Suppose it was in tape 1, we can so we can read the input and then copy, let us say, some parts into second and third tapes or however many tapes there are. Let us say we got to this situation. So let us say right now we are in this situation.

So, the a’s are in tape 1, b’s are in tape 2, c’s are in tape 3. So, now we can operate this Turing machine the following manner, you strike of one a from here and for one b here one c one b here one c here and so on. And we can work in a manner similar to the previous algorithm that we had earlier described. But because it is in separate tapes, it becomes simpler, because earlier we had the entire thing in one tape. So, you had to come back to the right then see the a’s, then go back to the left to cross off the c’s and so on. But when we have things in separate tapes, this going back and forth can be avoided.

So, this can result in some time savings as well. So, for each a we strike a b, we strike a c, we strike a b this, so the algorithm is pretty much the same. At the end, we uncross the b's and then again we strike the next a sorry, we strike the next a and then again strike a b, strike a c, strike a b, strike a c and so on. And if at any point b's remain, but c's do not remain we reject. If at any point b’s are struck out but c’s… or if everything else is struck out but c's remain even then we reject.

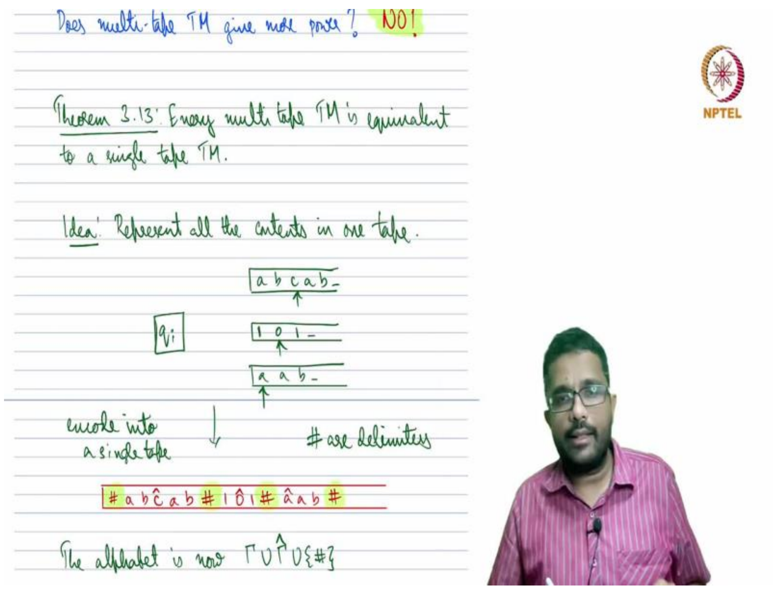
So, these are the 2 cases where we reject. If at the end c’s are remaining you reject. And if we do not have c's, we have struck off b but we are looking for c's but those are not there even then you reject. Otherwise, if everything matches up at the end you accept this. I am not elaborating on it because we have seen the similar, same language in the previous lecture. So, the algorithm is the same, just that this helps us avoid this multiple back and forth. Because it is in different places, different tapes, so it is organized better.

So, in the 3 tape machine, it is easier to check if the input is of this form $a^i b^j c^k$ where $k = i.j$. So, mostly what we see here is that there is some convenience that is afforded. Now, the natural question is, suppose we have 3 tapes, or suppose we have 10 tapes, suppose we have a big number of tapes. Clearly, whatever we do, you can just work it as a single tape machine. Because you could just completely decide to not use any of the remaining tapes and work everything in the first tape.

So what I mean to say is if you have a 10 tape machine, I could just focus on one tape and ignore the remaining space. What I am trying to say is that at k tape machine is at least as powerful as a 1 tape machine. A 10 tape machine is at least as powerful as a 1 tape machine.

Natural question to ask is, does it give us more power? Meaning are there languages that k tape machine can recognize that a one tape machine cannot? Or a 3 tape machine can recognize that a one tape machine cannot? And the answer is, as I have written in this slide, the answer is no. Meaning k tape model seems more powerful, but whatever a k tape machine can do, a 1 tape machine can also do.

(Refer Slide Time: 13:41)



Does multi-tape TM give more power? NO!

Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

q_i

a b c a b _

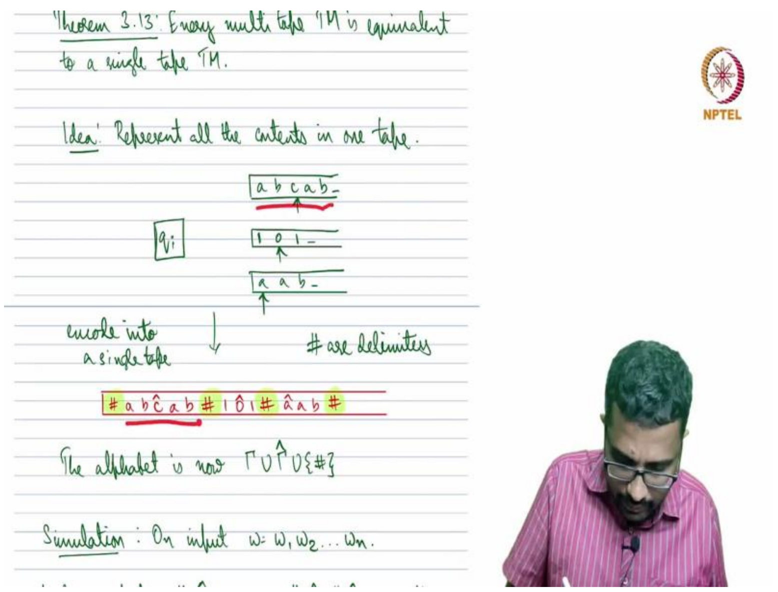
1 0 1 _

a a b _

encode into a single tape $\downarrow$ # are delimiters

$\# \; a \; b \; \hat{c} \; a \; b \; \# \; 1 \; \hat{0} \; 1 \; \# \; \hat{a} \; a \; b \; \#$

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

q_i

a b c a b _

1 0 1 _

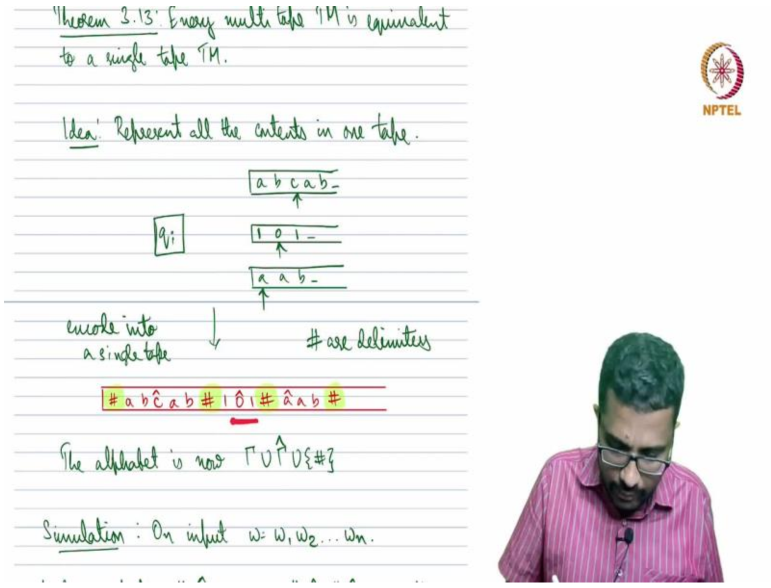
a a b _

encode into a single tape $\downarrow$ # are delimiters

$\# \; a \; b \; \hat{c} \; a \; b \; \# \; 1 \; \hat{0} \; 1 \; \# \; \hat{a} \; a \; b \; \#$

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \dots w_n$.



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

q_i | a b c a b _ | 1 0 1 _ | a a b _

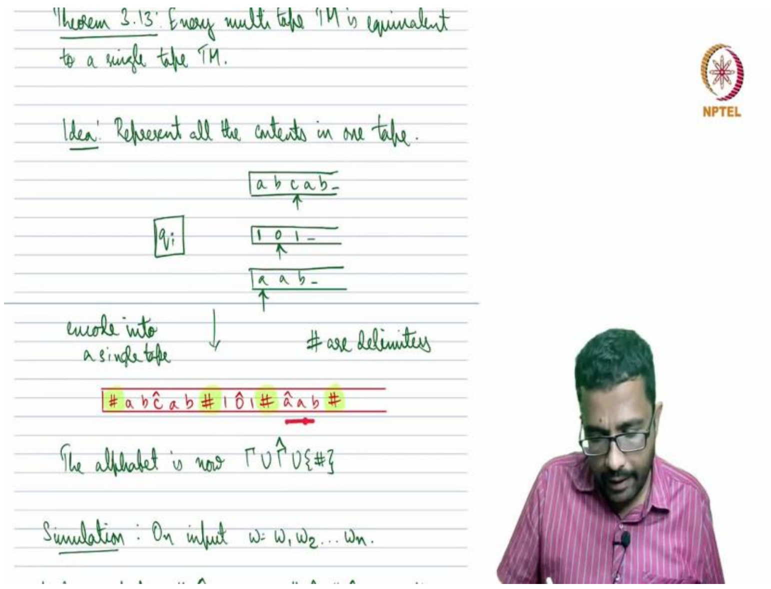
encode into a single tape

# are delimiters

# a b $\hat{c}$ a b \# 1 $\hat{0}$ 1 \# $\hat{a}$ a b \#

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

q_i | a b c a b _ | 1 0 1 _ | a a b _

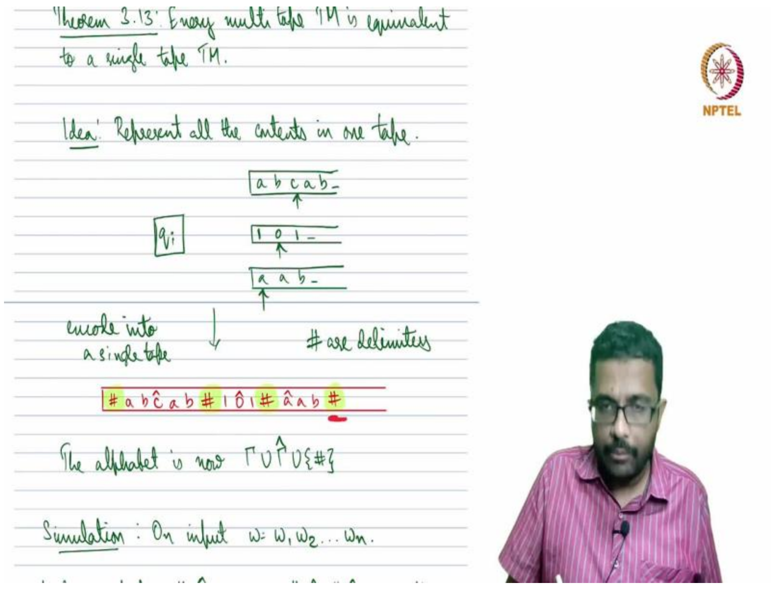
encode into a single tape

# are delimiters

# a b $\hat{c}$ a b \# 1 $\hat{0}$ 1 \# $\hat{a}$ a b \#

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

q_i | a b c a b _ | 1 0 1 _ | a a b _

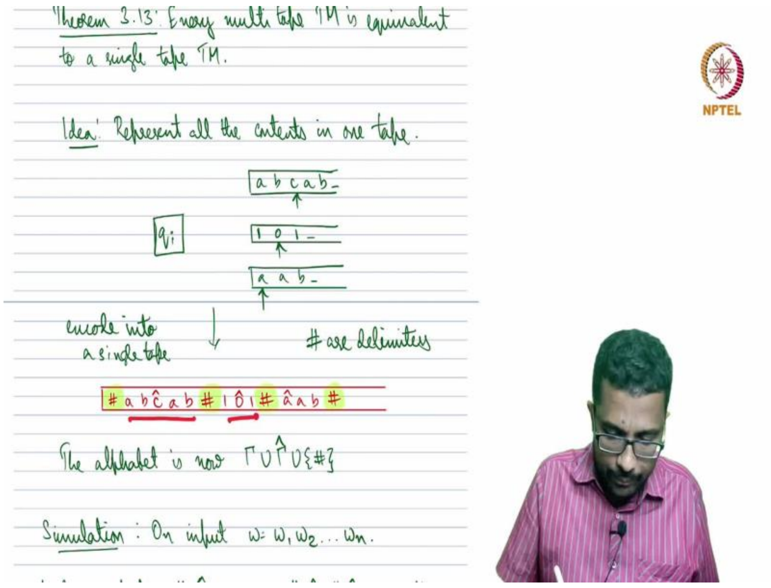
encode into a single tape

# are delimiters

# a b $\hat{c}$ a b \# 1 $\hat{0}$ 1 \# $\hat{a}$ a b \#

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

$\boxed{q_i}$

a b c a b _

1 0 1 _

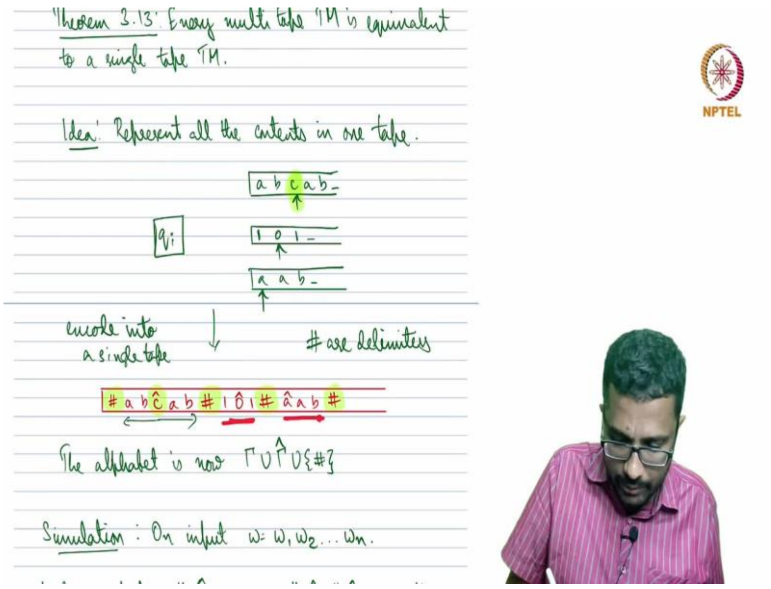
a a b _

encode into a single tape $\downarrow$ # are delimiters

a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

$\boxed{q_i}$

a b c a b _

1 0 1 _

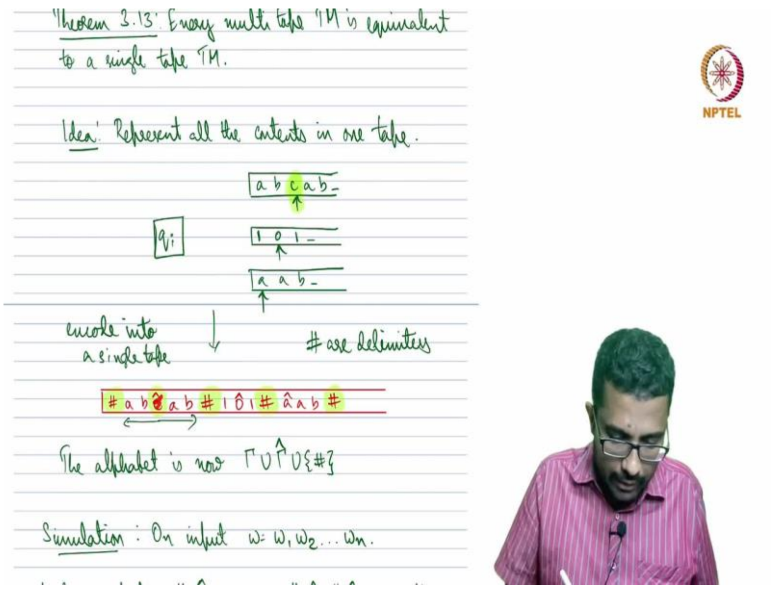
a a b _

encode into a single tape $\downarrow$ # are delimiters

a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

$\boxed{q_i}$

a b c a b _

1 0 1 _

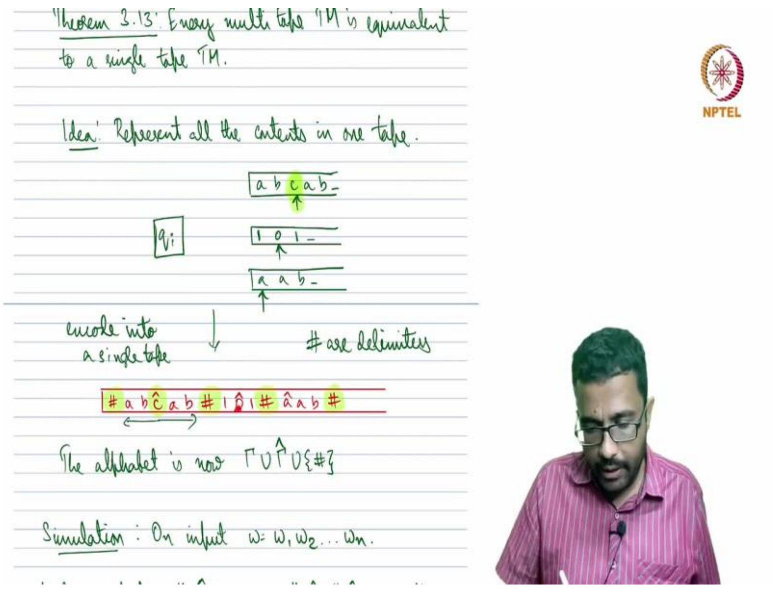
a a b _

encode into a single tape $\downarrow$ # are delimiters

a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.


Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

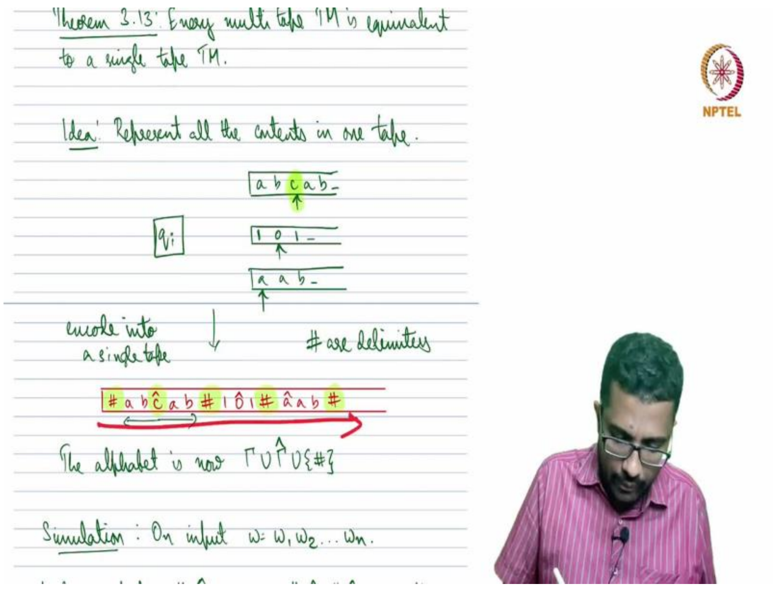
encode into a single tape

# are delimiters

$\# \, a \, b \, \hat{c} \, a \, b \, \# \, 1 \, \hat{0} \, 1 \, \# \, \hat{a} \, a \, b \, \#$

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.


Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

encode into a single tape

# are delimiters

$\# \, a \, b \, \hat{c} \, a \, b \, \# \, 1 \, \hat{0} \, 1 \, \# \, \hat{a} \, a \, b \, \#$

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.

Every multi tape machine is equivalent to a single tape machine. And the idea is fairly simple. So, let us see what the idea is. The idea is basically if you have k tapes, you try to compress all that information into one tape, basically encode. So, you write everything in one tape. this naturally means that you will have to do more back and forth which is unavoidable. But that is pretty much all that you have to do. So let us see.

Suppose I just pictorially represent a 3 tape machine. So, you have q_i being the state and there are 3 tapes. The first tape contains a b c a b the second tape contains 1 0 1 and the third tape contains a a b. For the first tape the head is at the third location, for the second tape at the second location, for the third tape at the first location. So, what you can do is to encode it in the following way. So basically, we encode the entire thing into a single tape, and these hash symbols are delimiters. So, hash are the delimiters meaning these symbols that I am going to highlight just now are going to separate or tell us where each tape content is ending and the next tape content is starting.

So, the first tape content that you see here is a b c a b and this is encoded between the first 2 hash symbols. After the second hash symbol, the content of the second tape starts. It is 1 0 1. After the third hash symbol, it is a a b, which is the content of the third tape and finally, we include one more hash to show that that is the end of the contents of the third tape. So, that is how we encode the contents. But there is also one more issue the 3 tape machine has 3 heads, whereas a one tape machine has only one head.

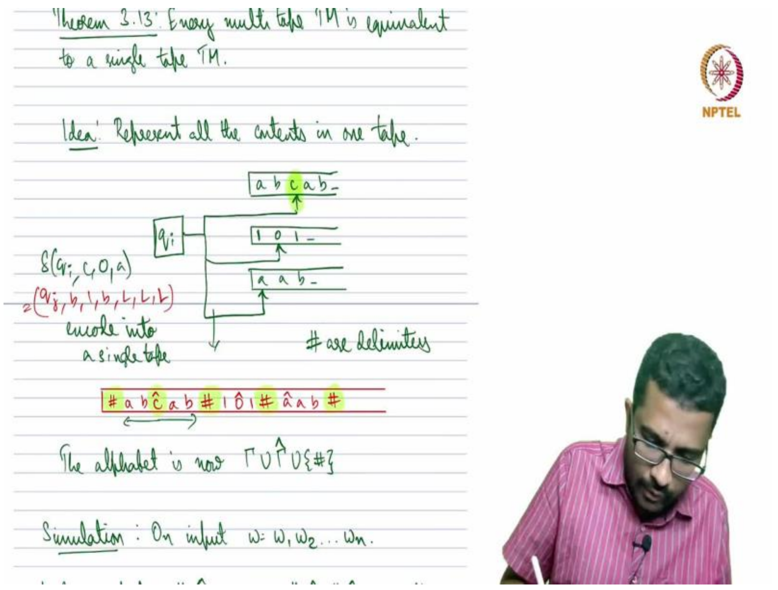
So, we also have to find some way to kind of keep track of the various head movements. The 3 tape machines had 3 heads, but a one tape machine has only one head. How to encode the head positions? What we do is we denote it in the tape itself. We have special symbols to kind of indicate the head position. So here, the third symbol c contains the head or the head is pointing to the third symbol of the first tape. I denote it by $\hat{c}$, with the hat, in the single tape conversion.

So, there is only one symbol in the part that corresponds to the first tape, this is the part that corresponds to the first tape, there is only one symbol with a hat. And that symbol indicates where the tape head was in the 3 tape equivalent. And similarly, we have one symbol with the hat in the second tape, which is $\hat{0}$, and one symbol with the hat in the third tape, $\hat{a}$, which is the first a. Now our alphabet size has at least doubled. I need all the original alphabet, all the things that were originally in the alphabet, which is Γ.

The new alphabet also needs every symbol with a hat on it. So now it is $\hat{c}$, but the next time when the tape head moves to the right, I may need $\hat{a}$. Similarly, second tape has $\hat{0}$ and the next time I may need $\hat{1}$. So, for every symbol, I also need the equivalent with a hat and also I need the delimiters hash. So, if the original alphabet was of size 10, now I need an alphabet of size 21. But that is all you need. It is not really a huge blow up.

So, now this is the situation of the of the Turing machine. You represent everything into one tape. And then what you do is you move from left to right. And you have only k tapes, in this case 3 tapes. So, you notice that in the first tape in the third location, the tape head was pointing at c. In the second tape, the tape head was pointing at 0. You know that because of a special symbol with the hat. And the third tape has the tape head pointing at a and then when you come to the end, you know you are at the end. So, now you know that the first tape head looked at a c, second tape looked at 0, third tape head looked at an a. And you know that the state was q_i.

(Refer Slide Time: 18:32)



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

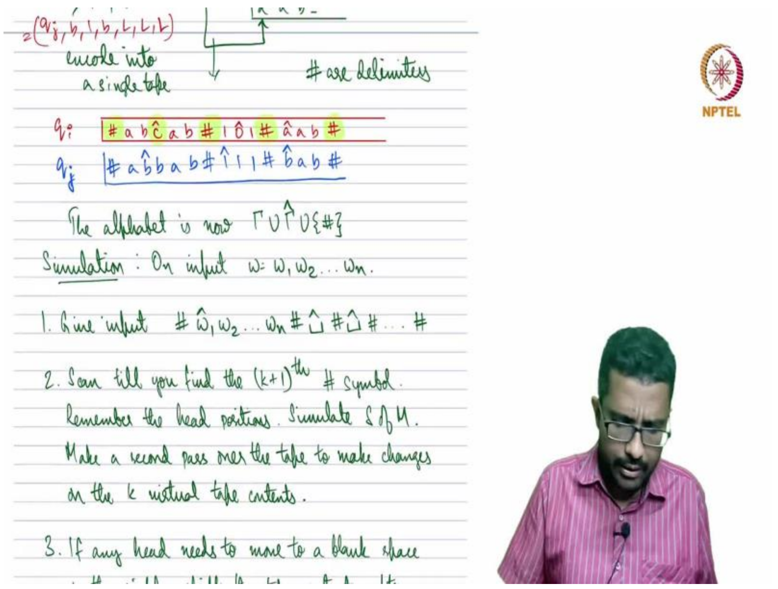
$\delta(q_i, c, 0, a)$
$= (q_j, b, 1, b, L, L, L)$
encode into a single tape

are delimiters

a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation : On input $w = w_1 w_2 \ldots w_n$.



$= (q_j, b, 1, b, L, L, L)$
encode into a single tape

are delimiters

q_i # a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b #

q_j # a $\hat{b}$ b a b # $\hat{1}$ 1 1 # $\hat{b}$ a b #

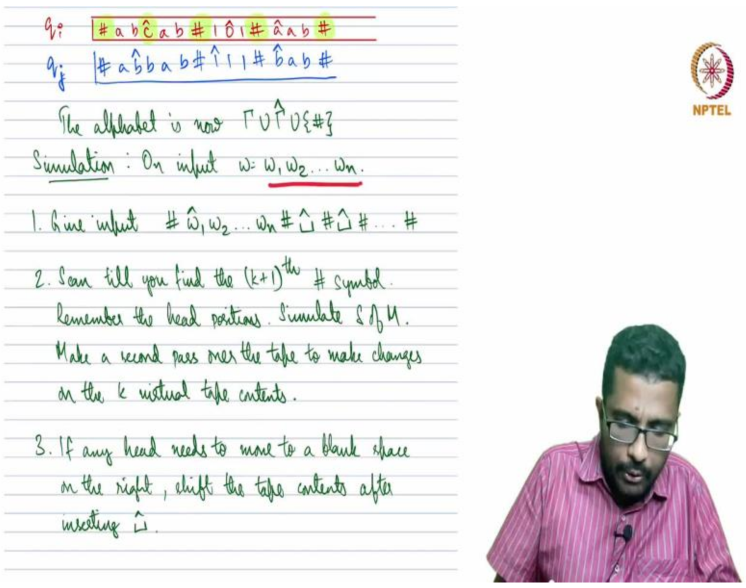
The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation : On input $w = w_1 w_2 \ldots w_n$.

1. Give input $\# \hat{w}_1 w_2 \ldots w_n \# \hat{⌴} \# \hat{⌴} \# \ldots \#$

2. Scan till you find the $(k+1)^{th}$ # symbol. Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.

3. If any head needs to move to a blank space



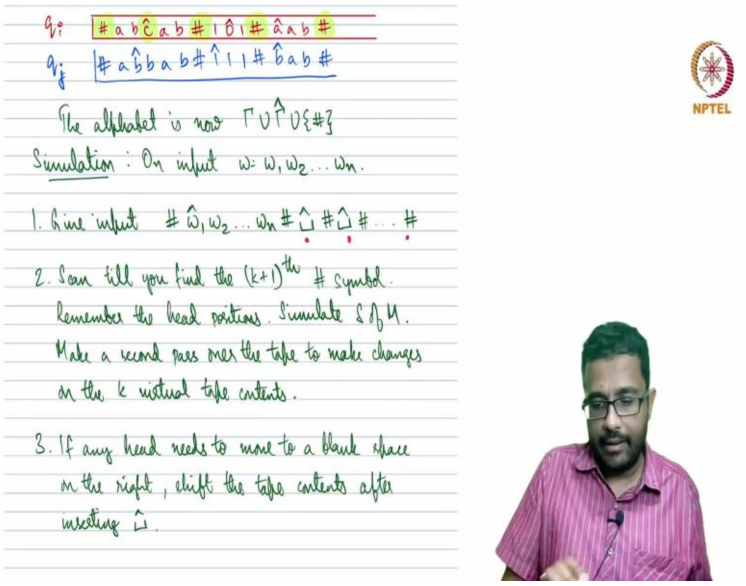
q_i: $|\# a b \hat{c} a b \# 1 \hat{0} 1 \# \hat{a} a b \#$

q_j: $|\# a \hat{b} b a b \# \hat{1} 1 1 \# \hat{b} a b \#$

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation : On input $w = w_1 w_2 \ldots w_n$.

1. Give input $\# \hat{w}_1 w_2 \ldots w_n \# \hat{\sqcup} \# \hat{\sqcup} \# \ldots \#$
2. Scan till you find the (k+1)th # symbol. Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.
3. If any head needs to move to a blank space on the right, shift the tape contents after inserting $\hat{\sqcup}$.



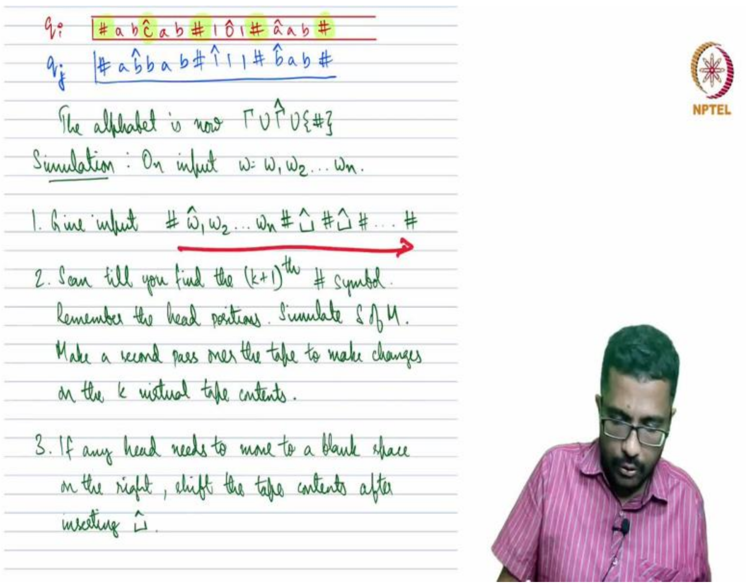
q_i: $|\# a b \hat{c} a b \# 1 \hat{0} 1 \# \hat{a} a b \#$

q_j: $|\# a \hat{b} b a b \# \hat{1} 1 1 \# \hat{b} a b \#$

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation : On input $w = w_1 w_2 \ldots w_n$.

1. Give input $\# \hat{w}_1 w_2 \ldots w_n \# \hat{\sqcup} \# \hat{\sqcup} \# \ldots \#$
2. Scan till you find the (k+1)th # symbol. Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.
3. If any head needs to move to a blank space on the right, shift the tape contents after inserting $\hat{\sqcup}$.



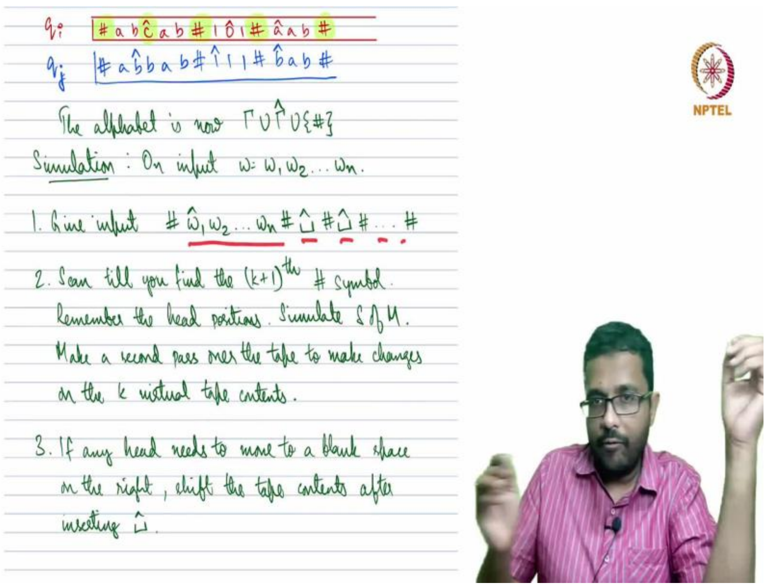
q_i: $|\# a b \hat{c} a b \# 1 \hat{0} 1 \# \hat{a} a b \#$

q_j: $|\# a \hat{b} b a b \# \hat{1} 1 1 \# \hat{b} a b \#$

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation : On input $w = w_1 w_2 \ldots w_n$.

1. Give input $\# \hat{w}_1 w_2 \ldots w_n \# \hat{\sqcup} \# \hat{\sqcup} \# \ldots \#$
2. Scan till you find the (k+1)th # symbol. Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.
3. If any head needs to move to a blank space on the right, shift the tape contents after inserting $\hat{\sqcup}$.



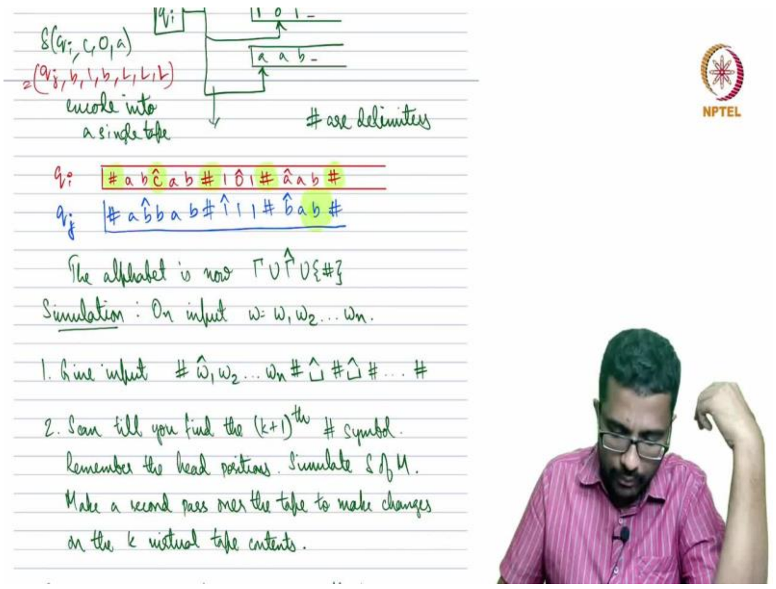
q_i # a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b #

q_j # a $\hat{b}$ b a b # $\hat{1}$ 1 1 # $\hat{b}$ a b #

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation : On input $w = w_1 w_2 \dots w_n$.

1. Give input $\# \underline{\hat{w}_1 w_2 \dots w_n} \# \hat{\sqcup} \# \hat{\sqcup} \# \dots \#$
2. Scan till you find the (k+1)th # symbol. Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.
3. If any head needs to move to a blank space on the right, shift the tape contents after inserting $\hat{\sqcup}$.



$\delta(q_i, c, 0, a)$
$= (q_j, b, 1, b, L, L, L)$
encode into a single tape

are delimiters

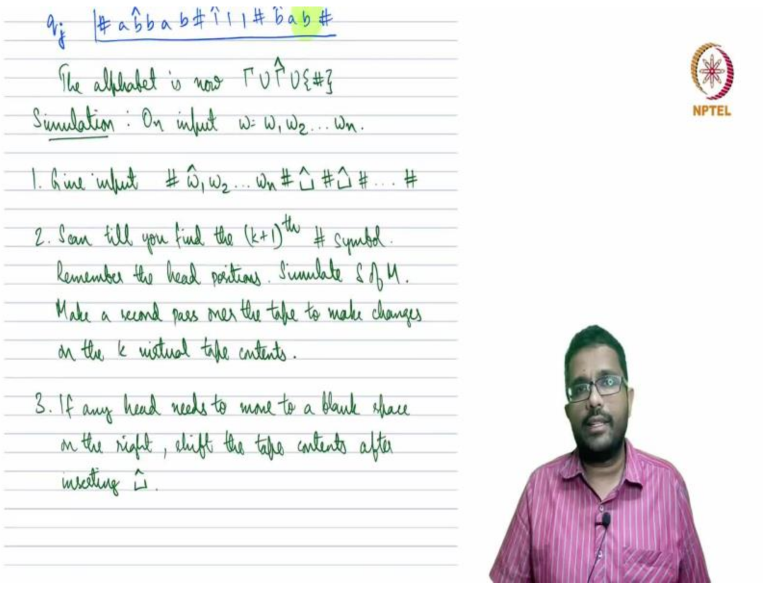
q_i # a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b #

q_j # a $\hat{b}$ b a b # $\hat{1}$ 1 1 # $\hat{b}$ a b #

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation : On input $w = w_1 w_2 \dots w_n$.

1. Give input $\# \hat{w}_1 w_2 \dots w_n \# \hat{\sqcup} \# \hat{\sqcup} \# \dots \#$
2. Scan till you find the (k+1)th # symbol. Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.

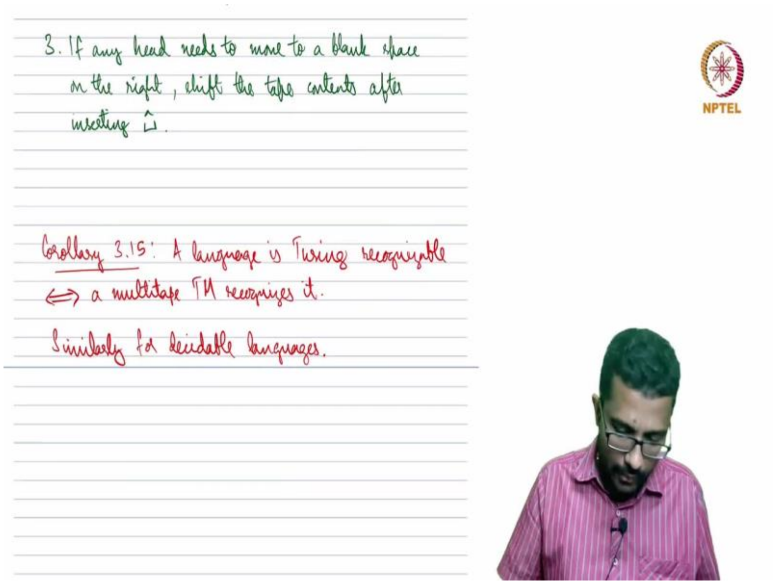


q_j # a $\hat{b}$ b a b # $\hat{1}$ 1 1 # $\hat{b}$ a b #

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation : On input $w = w_1 w_2 \dots w_n$.

1. Give input $\# \hat{w}_1 w_2 \dots w_n \# \hat{\sqcup} \# \hat{\sqcup} \# \dots \#$
2. Scan till you find the (k+1)th # symbol. Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.
3. If any head needs to move to a blank space on the right, shift the tape contents after inserting $\hat{\sqcup}$.



3. If any head needs to move to a blank space on the right, shift the tape contents after inserting $\hat{\sqcup}$.

Corollary 3.15: A language is Turing recognizable $\Longleftrightarrow$ a multitape TM recognizes it.

Similarly for decidable languages.

So, now you can look at the rule. You can just ask what was $\delta(q_i, c, 0, a)$. Now, suppose this asked us to go to let us say something like $(q_j, b, 1, b, \mathrm{L}, \mathrm{L}, \mathrm{L})$. So, suppose this is the rule of this tape. So, first what we do is the head goes from left to right, understands what is represented and comes back to the left. So now it knows that it is in state q_i and heads are pointing to $c, 0, a$.

Now it knows the rules of the original multi tape Turing machine. Which is now $\delta(q_i, c, 0, a) = (q_j, b, 1, b, \mathrm{L}, \mathrm{L}, \mathrm{L})$. So, this was q_i, which means now you go to state q_j.

$$q_i: \ \# \, a \, b \, \hat{c} \, a \, b \, \# \, 1 \, \hat{0} \, 1 \, \# \, \hat{a} \, a \, b \, \#$$
$$q_j: \ \# a \, \hat{b} \, b \, a \, b \, \# \, \hat{1} \, 1 \, 1 \, \# \, \hat{b} \, a \, b \, \#$$

So you have hash. At the first head, you write b here, so, $a \, \hat{b} \, b \, a \, b$ and then you move left. So, the head comes to the second position. In the second tape head, you write a 1. So, you have $\hat{1} \; 1 \; 1$, tape head moves left. And the third position, you have $\hat{b} \, a \, b$. You moved the head left, but then from the leftmost position you cannot move further left. So, this is what you have.

So, now, you know this is a rule and then you make another pass from left to right and you implement this change in the single tape. So, you move from left to right, you read where the heads are, then you come back to the left. You know the transition function of the original multi tape Turing machine. So, you know the δ thing. δ was $(q_j, b, 1, b, \mathrm{L}, \mathrm{L}, \mathrm{L})$. Now, you make another pass from left to right and implement this, which is what we have done.

Basically, what we are doing is we are just making these multiple passes and implementing this. So, for every step of the multi tape Turing machine, now I need to go, let us say, first left to right, then come back to the left, then again left to right, and come back to the left. So maybe 4 moves that span the entire tape length. Again, I have written the same thing here.

Originally the input is, let us say $w = w_1 w_2 \dots w_n$. The original starting configuration is, the entire input is in the first tape between the first and second hash, then every other tape is blank. Now, you scan it from left to right. So, you have k tapes. So, the k plus 1 hash symbol indicates the end of the tapes. You remember the head positions. Now, you know the rules of the multi tape Turing machine. You make a second pass, and you update this K virtual tape.

So, basically it is like a virtual memory. So you have a single tape, but you have these k virtual tape locations where you are encoding it. And suppose in this case, it did not really happen. But suppose at some stage, we need to make space. Let us say, in this tape position. Maybe I will highlight something.

Let us say this b or the third tape here. Let us say in the third tape here, I wanted to write b, a, b, b. I want to move one step to the right, so I need to move the hash and create space. But we do that if that is required, we move the hash symbol and create a blank. And that is how we would implement a multi tape Turing machine in a single tape. Basically, it is a simulation. But it increases the running time, as you may guess. Because earlier, you could move the things in the single tape left or right individually. But now every single move requires us to scan the entire tape from left to right, multiple times. So that is how you would implement a multi tape Turing machine using a single tape.

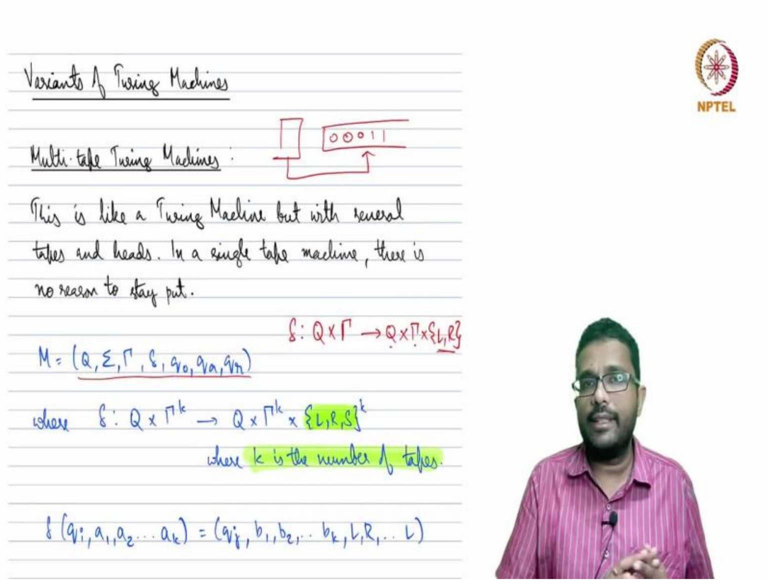
And what this tells us is that anything that you can do with a multi tape Turing machine can also be done with a single tape Turing machine. The moves, accept, reject, loop, everything exactly remains the same. So, we did not really change any functionality. Whatever input would have been accepted, would still remain accepted. Whatever was rejected would still be rejected. Whatever looped would still loop. Which means it is exactly equivalent.

So, if a language was Turing recognizable in a single tape model, it is also Turing recognizable in the multi tape model. And vice versa. It is if and only if. Because multi tape is at least as powerful as single tape, it is easy to see. In this proof we say that it is not any more powerful. So, the class of languages that are Turing recognizable by single tape is exactly the class of languages that are Turing recognizable by multi tape. So, nothing more nothing less.

Same for decidable. If a language was decidable in a multi tape Turing machine, it is decidable a single tape Turing machine as well. Same the other way also. So, let us see other models also. In all these cases, we will see that the definitions of recognizable and decidable. They are kind of robust. They do not really depend on these different models that we are defining. So, in that sense it is good.

So, we do not have to qualify that this language is decidable for single tape machine and not for multi tape. We do not have to do any of that. Recognizable means recognizable in single tape and multi tape, or not recognizable in either model. It is same for decidable.

(Refer Slide Time: 25:37)



Variants of Turing Machines

Multi-tape Turing Machines :

This is like a Turing Machine but with several tapes and heads. In a single tape machine, there is no reason to stay put.

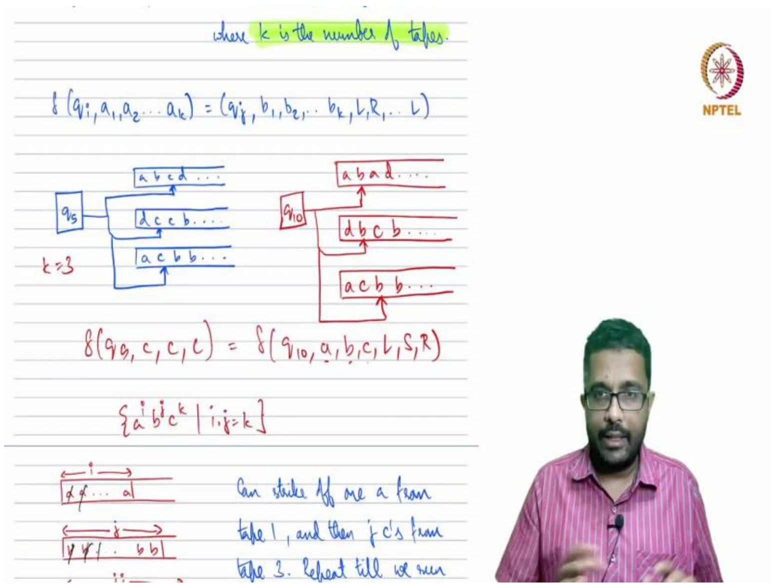
$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

where $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

where k is the number of tapes.

$\delta(q_i, a_1, a_2 \ldots a_k) = (q_j, b_1, b_2, \ldots b_k, L, R, \ldots L)$



where k is the number of tapes.

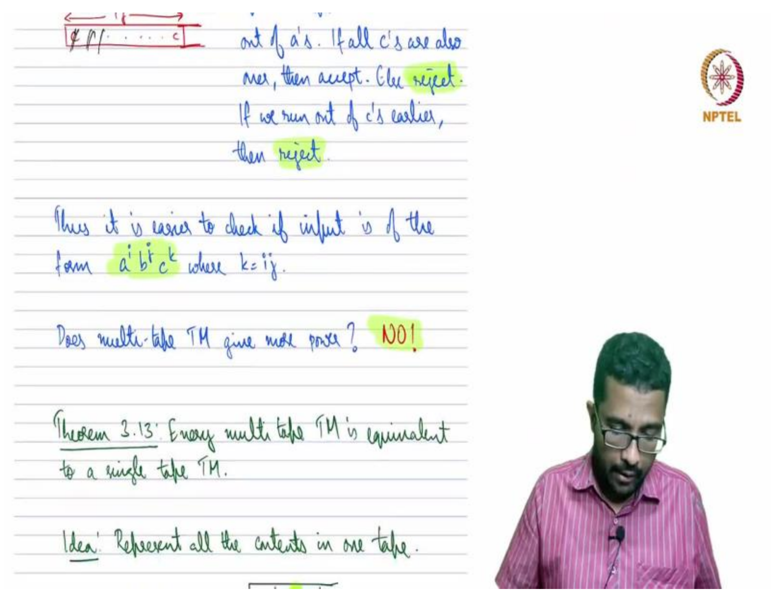
$\delta(q_i, a_1, a_2 \dots a_k) = (q_j, b_1, b_2, \dots b_k, L, R, \dots L)$

$\delta(q_5, c, c, c) = \delta(q_{10}, a, b, c, L, S, R)$

$\{a^i b^j c^k \mid i.j = k\}$

i
$\cancel{a}\,\cancel{a}$ . . . a
j
$\cancel{b}\,\cancel{b}$ / . b b

Can strike off one a from tape 1, and then j c's from tape 3. Repeat till we run



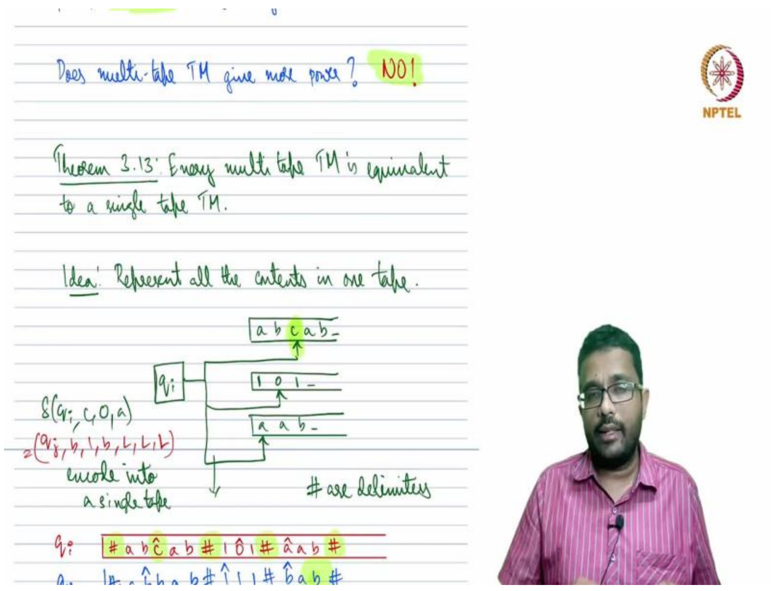
out of a's. If all c's are also over, then accept. Else reject. If we run out of c's earlier, then reject.

Thus it is easier to check if input is of the form $a^i b^j c^k$ where $k = ij$.

Does multi-tape TM give more power? NO!

Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.



Does multi-tape TM give more power? NO!

Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

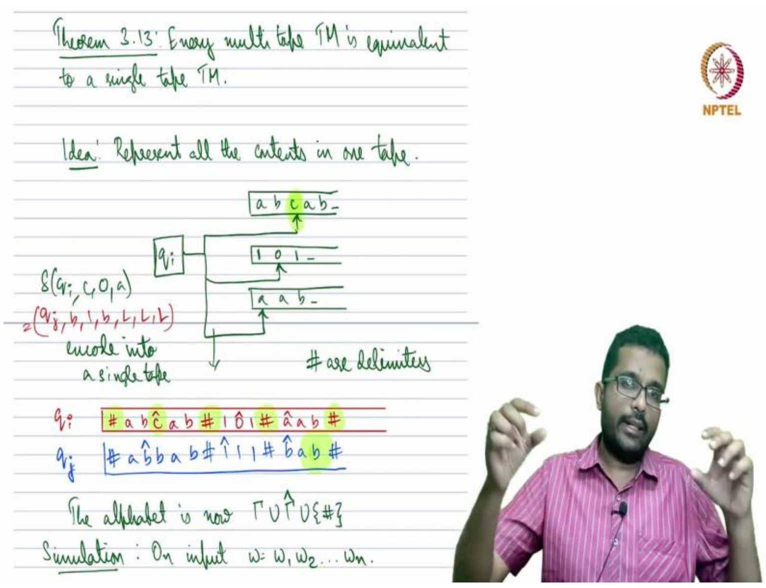
$\delta(q_i, c, 0, a)$
$= (q_j, b, 1, b, L, L, L)$

encode into a single tape

are delimiters

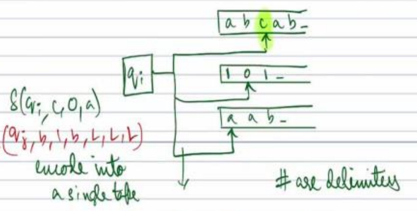
q_i: # a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b #

q_j: # a $\hat{b}$ b a b # $\hat{1}$ 1 1 # $\hat{b}$ a b #



Theorem 3.13: Every multi tape TM is equivalent to a single tape TM.

Idea: Represent all the contents in one tape.

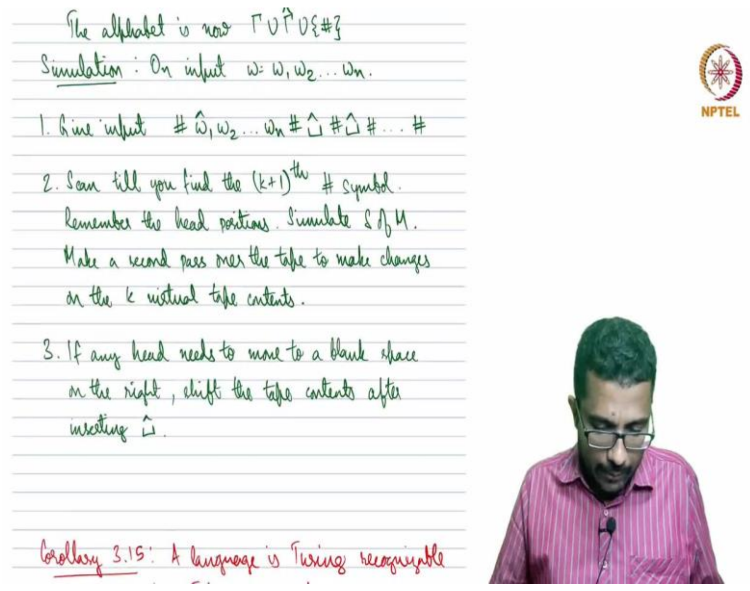


q_i: | # a b $\hat{c}$ a b # 1 $\hat{0}$ 1 # $\hat{a}$ a b # |

q_j: | # a $\hat{b}$ b a b # $\hat{1}$ 1 1 # $\hat{b}$ a b # |

The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

Simulation: On input $w = w_1 w_2 \ldots w_n$.



The alphabet is now $\Gamma \cup \hat{\Gamma} \cup \{\#\}$

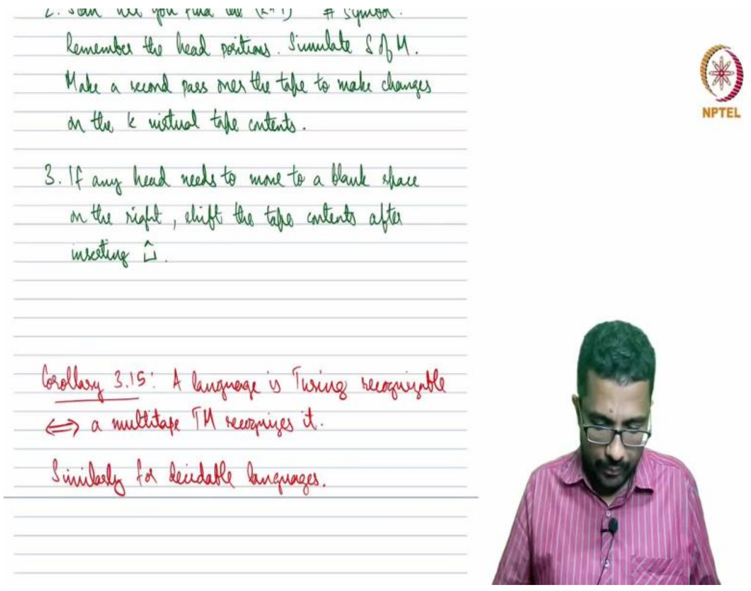
Simulation: On input $w = w_1 w_2 \ldots w_n$.

1. Give input $\# \hat{w}_1 w_2 \ldots w_n \# \hat{\sqcup} \# \hat{\sqcup} \# \ldots \#$

2. Scan till you find the $(k+1)^{th}$ # symbol. Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.

3. If any head needs to move to a blank space on the right, shift the tape contents after inserting $\hat{\sqcup}$.

Corollary 3.15: A language is Turing recognizable



Remember the head positions. Simulate δ of M. Make a second pass over the tape to make changes on the k virtual tape contents.

3. If any head needs to move to a blank space on the right, shift the tape contents after inserting $\hat{\sqcup}$.

Corollary 3.15: A language is Turing recognizable $\iff$ a multitape TM recognizes it.

Similarly for decidable languages.

And that completes this lecture, that is lecture number 29. We saw the multi tape Turing Machine model. And we saw how it computes. We saw that it is not any more powerful than a single tape Turing machine. So, the idea of this proof was that you can encode the contents of multiple tapes into a single tape using these hash delimiters and hat symbols. We can kind of create virtual tapes in the single tape itself.

And by sacrificing on the time by making lots more moves left, right, left, right, left to the right, we are able to encode or we are able to reflect whatever happens in the multi tape Turing machine in the single tape Turing machine. So that completes lecture 29. In the next lecture, we will see another variant of Turing machine which also will turn out to be equivalent, which will be the non-deterministic Turing machine. So that is it for lecture 29 and see you in lecture 30. Thank you.