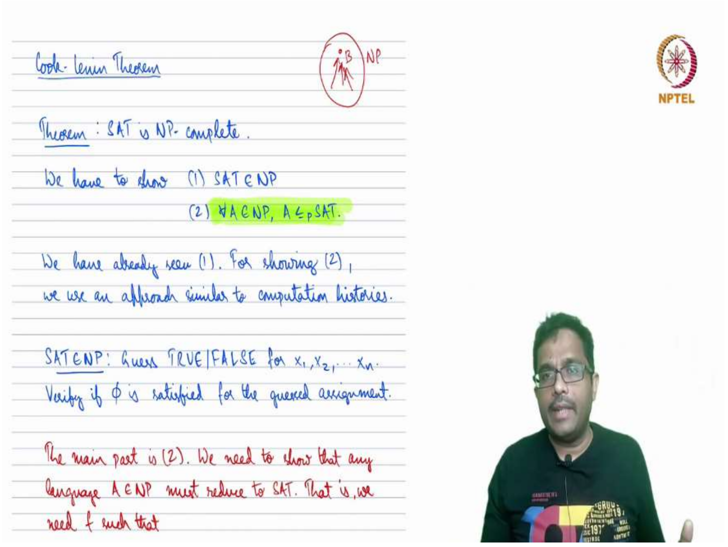
Theory of Computation

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

Cook-Levin Theorem

(Refer Slide Time: 0:16)



Cook-Levin Theorem

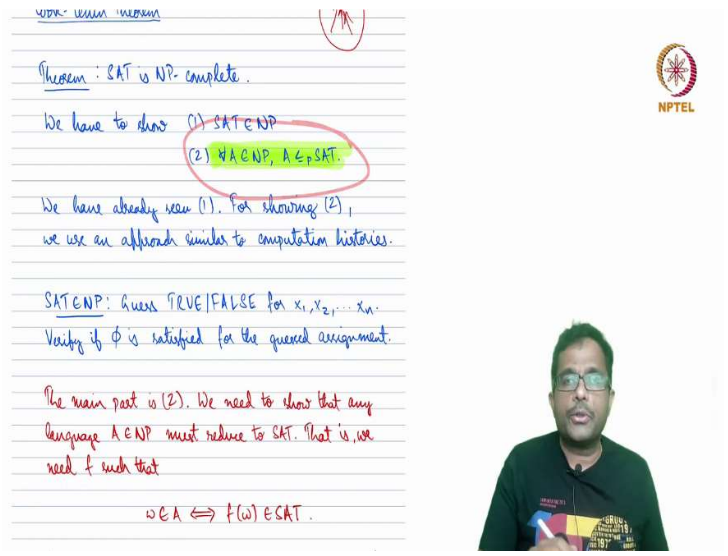
Theorem: SAT is NP-complete.

We have to show (1) SAT $\in$ NP
(2) $\forall A \in NP, A \leq_p SAT$.

We have already seen (1). For showing (2), we use an approach similar to computation histories.

SAT $\in$ NP: Guess TRUE/FALSE for $x_1, x_2, \dots x_n$. Verify if ϕ is satisfied for the guessed assignment.

The main part is (2). We need to show that any language $A \in NP$ must reduce to SAT. That is, we need f such that



$$w \in A \iff f(w) \in SAT.$$

Hello and welcome to lecture 50 of the course theory of computation. This is going to be a very important lecture in the course, because we are going to be showing one of the important results, one of the most important lessons that we will be learning in the course. So, in this lecture, we will see Cook-Levin theorem, which shows that SAT is NP-complete.

In the previous lecture, we defined NP-completeness, and then we discussed how is it possible to show that something is NP-complete. So, to show that a certain language is NP-complete, we have to show that all the languages, let us say if you have to show the B is NP-complete, we have to show all the other languages in NP are reducible to it. Fortunately, if we have some NP-complete language, let us say, if let us say B is known to be an NP-complete language, to show that C is NP-complete, it is enough to reduce B to C.

But to begin with, we need to have some already known NP-complete language. So, Cook-Levin theorem provides us with that known NP-complete language, and that is satisfiability. So, we have to show two things in order to show that SAT is NP-complete. First is that SAT is in NP. Second is that all the languages in NP are going to be reducible to SAT.

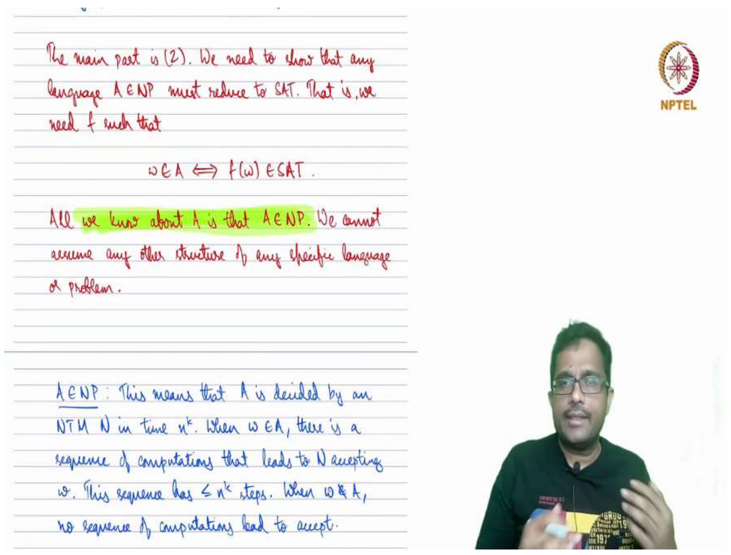
The second step is going to be the hard step. The first step is actually something that in fact, we have already said in the course, in one of the previous lectures. Anyway, it is brief, I will quickly explain again. The second step is going to be the hardest step. So, in order to show that all the languages in NP are reducible to SAT.

So, let me just quickly recap how to show that SAT is in NP. So, we just have a guess and verify Turing machine. So, we guess so, let us say SAT has the Boolean formula, that is given φ has n variables let us say $x1$ to xn. So, we non-deterministically guess true, false values for all of these n variables, and then after guessing the n variables, so and then you verify whether this assignment that is guessed, does it satisfy the formula phi or not?

If the assignment satisfies the formula phi, you accept. If it does not satisfy, we reject. So, if there is a satisfying assignment, one of these 2^n guesses will be the satisfying assignment, and then we will accept. If it is not, if the formula is not satisfiable, none of these guesses will lead to a satisfying assignment, because the formula is not satisfied.

Hence, all the parts will lead to reject, because whatever you try out, it is not going to work. So, that is how we would show that SAT is in NP. So, now, let us focus on the second part of the proof, to show that all languages in NP are reducible to SAT.

(Refer Slide Time: 3:26)

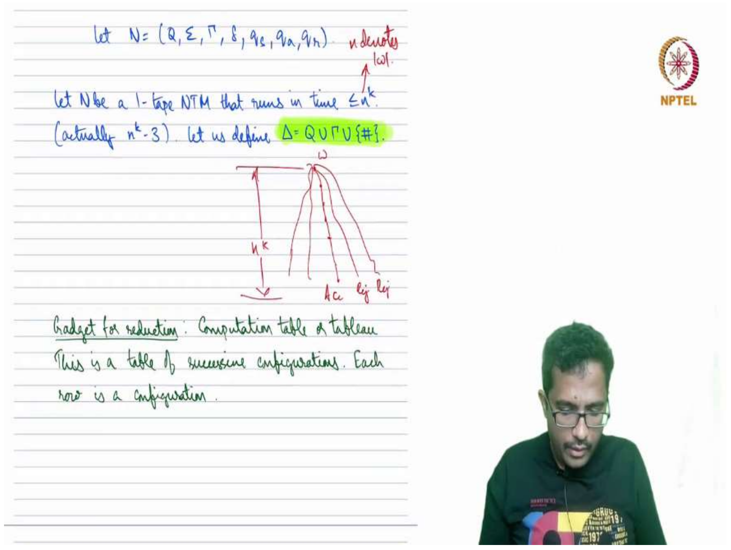


The main part is (2). We need to show that any language $A \in NP$ must reduce to SAT. That is, we need f such that

$$w \in A \iff f(w) \in SAT.$$

All we know about A is that $A \in NP$. We cannot assume any other structure of any specific language or problem.

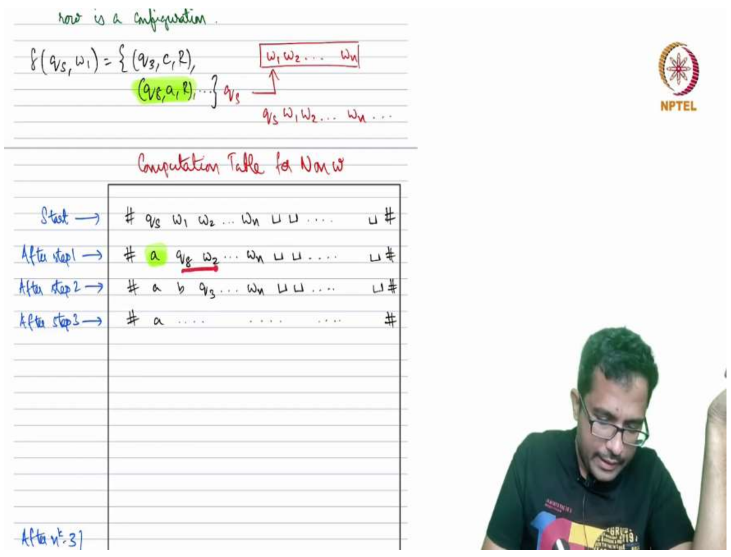
$A \in NP$: This means that A is decided by an NTM N in time n^k. When $w \in A$, there is a sequence of computations that leads to N accepting w. This sequence has $\leq n^k$ steps. When $w \notin A$, no sequence of computations lead to accept.



Let $N = (Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$. n denotes $|w|$.

Let N be a 1-tape NTM that runs in time $\leq n^k$. (actually $n^k - 3$). Let us define $\Delta = Q \cup \Gamma \cup \{\#\}$.

Gadget for reduction: Computation table or tableau
This is a table of successive configurations. Each row is a configuration.



row is a configuration.

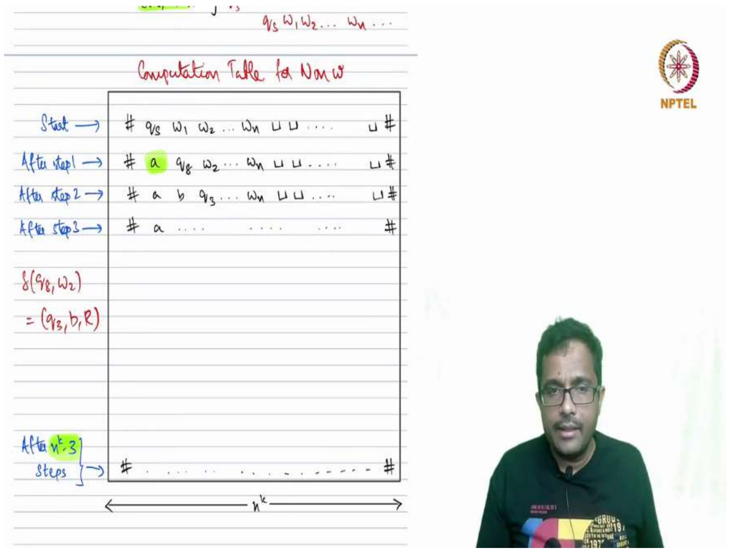
$$\delta(q_s, w_1) = \{(q_3, c, R), (q_8, a, R), \dots\}$$

q_s $\boxed{w_1 w_2 \dots w_n}$

$q_s\, w_1 w_2 \dots w_n \dots$

Computation Table for N on w

Start $\rightarrow$	$\#\ q_s\ w_1\ w_2 \dots w_n\ \sqcup\ \sqcup \dots\ \sqcup\ \#$
After step 1 $\rightarrow$	$\#\ a\ q_8\ w_2 \dots w_n\ \sqcup\ \sqcup \dots\ \sqcup\ \#$
After step 2 $\rightarrow$	$\#\ a\ b\ q_3 \dots w_n\ \sqcup\ \sqcup \dots\ \sqcup\ \#$
After step 3 $\rightarrow$	$\#\ a \dots \dots \dots\ \#$
After $n^k - 3$	


$q_s w_1 w_2 \dots w_n \dots$
Computation Table for N on w
Start → # q_s w_1 w_2 … w_n ⊔ ⊔ …. ⊔ #
After step 1 → # a q_8 w_2 … w_n ⊔ ⊔ …. ⊔ #
After step 2 → # a b q_3 … w_n ⊔ ⊔ … ⊔ #
After step 3 → # a …. …. …. #
$\delta(q_8, w_2) = (q_3, b, R)$
After $n^k - 3$ steps → # …. #
n^k
NPTEL

So, the thing is, we saw one reduction in one of the previous lectures, we saw 3-SAT reduces to CLIQUE, but then we knew that it is a 3-SAT, and we have to get a CLIQUE instance. But here we have to deal with some abstract language A, and A, we have to show that a reduces to SAT, the only thing that we know about A is that A is in NP, this is only thing that we know about it, and we have to show that this language A reduces to SAT.

So, if we knew something about the language A, then we can use the structure of the language, or we can use that to build the reduction, but here we have no information about A. The only information that we have is that A is in NP. So, only information that we have about A is that A is in NP.

Now, this is the challenge here, we have to build a reduction from this language A, about which, we do not know much about, such that if a string is in A, the reduced instance, the $f(w)$ should be in SAT, and if the string is not in A, $f(w)$ should not be in SAT. So, we have to somehow build a Boolean formula from w, so as to get this connection, so as you get this relation.

So, that is the challenge of this, and that is why this is an important result. So let us try to understand it, let us try to see what we have? So, the only thing that we have is A is in NP. So, now what does it mean? Let us try to understand this. This is pretty much the only thing that we have, and we have to work with this. So, what is A in NP mean?

A is in NP means that A has a polynomial time non deterministic Turing machine that decides it, or non-deterministic polynomial time decider. So, let us say A is decided by a non-deterministic Turing machine N, this is just the definition of NP. In polynomial time, so, to be a bit more specific let us say it gets decided in time n^k.

So, this means that. So, there is a non-deterministic Turing machine that runs in time n^k decides, this means that a string in A, w is input into this Turing machine, then we have a sequence of computations. So, there could be many computation paths.

Let us say w is input to N, there could be many computation paths, but we know at least one of them is an accepting computation path. We know at least one of them is an accepting, so, the others could be reject, but we know at least one accepting path exists. So, what does it mean? One accepting path means there is a sequence of configurations.

So, starting with the starting configuration, then another configuration, and then another configuration and so on, there is a valid sequence of configurations that lead to an accepting state, or accepting configuration. So, I am just, we are just analysing what it means to say that is A is in NP.

So, when there is a string that is in A, there is a sequence of computations that leads N to an accepting computation. Meaning, starting from the starting configuration through a series of valid successor configuration, finally reaching to accepting configuration, and we know that N decides A in n^k time, so, the number of steps here is at most n^k.

So, which means number of steps here is at most n^k. So, there is a sequence of computations, starting with the starting configuration, and reaching to an accept for N on the string w. And when w is not in A. So, this is when w is in A. When w is not in A, there is no such sequence of computations, meaning whatever valid sequence of computations or configurations that we see, we need to reject, that is what it means.

So, when w is in A, there is a sequence of computations leading to accept, and w is not in A, all the sequence of computations need to reject. So, now, let us define a bit more. So, this is what we have, and this is what we will work with. Now, let us define a bit more about N. So, let N be the following Q, sigma, gamma, delta, q start, q accept, and q reject. So, this is a standard notation.

$$N = (Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$$

Q is a set of states, Σ is the input alphabet, Γ is the tape alphabet, δ is the transition function. q_s the starting state, q_a is accepting state, and q_r is the rejecting state. And further we will assume that n is a one tape non-deterministic Turing machine. And let us say it runs in time n^k, where n denotes the length of the input.

And in fact, to be bit more precise, we assume that it runs in $n^k - 3$, which is not really big assumption because if it needs n^k then we will go to the next k, we will go to n^{k+1}, so that it runs in $n^{k+1} - 3$. And we define delta to be the set of all the set of states, tape alphabet, and the delimiter symbol hash.

$$\Delta = Q \cup \Gamma \cup \{\#\}$$

So, we define delta to be this, maybe I will highlight this. So basically, what is delta? Delta is any symbol that shows up in a configuration. So, if you remember configuration had this notation, where we have the tape content, and we had inserted the state into the tape content.

So, for instance, the starting configuration is say $w1, w2,$ up to wn, with the starting state, and this is denoted by q_s, because the head is in the first position, followed by the input. And this is the starting configuration, and then just to indicate start and end, we put a delimiter, we use a delimiter hash at the beginning and end.

So, this is the role of the three things that we have in delta. One is the state? One is the tape alphabet. State and tape alphabet we need to denote the configuration and the symbol hash serves as a delimiter. So now we need to reduce A, the language A of which the only thing that we know is that it has an NTM decider N, which runs in n^k.

We want to reduce it to SAT. So, we will have a gadget, we need to reduce it. And the gadget that is used for the reduction is a computation table. It is called a computation table. Sometimes it is called computation tableau, depending on which book you refer. Basically, it is just a table listing successive configurations. So, I have written down here, this is the computation table, maybe I will just write down. For N on w.

So basically each row is a configuration. So to start with, this is the starting configuration, where we have the input tape contains only the input $w1$ to wn, n symbols, and the head is pointing to $w1$. That is why the state is the start state, that is why the start state is written here, followed by a bunch of blanks. How many blanks? We will come to that in a moment, and then we have a delimiter hash, at the beginning and end.

And how many blanks? So basically, what we do is , we said that it decides in n^k. So in n^k time, the tape can, the tape head can move up to the n^k th position. So farthest the tape head can move is one step right, each step. So each step it goes one step right. So, that is the farthest that the tape head can move.

So, the maximum that it can cover in the tape is n^k locations. So, the rest of the locations, we put blanks. So, that is what I have indicated at the bottom of the screen, n^k is the width of this table, or tableau. So, the first n locations, we put $w1$ to wn. In fact, before that we have the hash and the q_s, followed by blanks, and the last location is a hash again.

Now, this is a computation table. Now, let us say one of the possible moves of the Turing machine, of the NTM N was this. Let us say delta. So initially, you are the starting state, and you are reading the symbol w1. Now, there could be multiple options available. There could be q3, C, right. There could be q8, a, right. There could be multiple options available.

There could be multiple options available because it is a non-deterministic Turing machine. There could be multiple options available, let us take one option, let us take the Turing machine opted for this option, it moves to state q8, it writes a in the location where w1 was, and then it moves one step to the right.

So that corresponds to the second configuration written here. So, it writes a. So a is in the first location, in place of w1, and then the tape head moves right. That is why the q8 has, the state q8 is written in the second location now, because now the tape head, after this move, is pointing to the second symbol. And suppose there is another move based on this let us say delta, based on the second step let us say, we have $\delta(q8, w2)$.

So, it is pointing to w2 and in state q8. Suppose the move was q3, b, and right. There could be multiple moves suppose this was a selected move, then we see the move in the next line. So, now, this w2 is replaced by b, and the state is q3, and the state moves one step to the right. So, this is going to be one possible configuration that the Turing machine can get to after two steps. So, like that we can keep filling up the computation table.

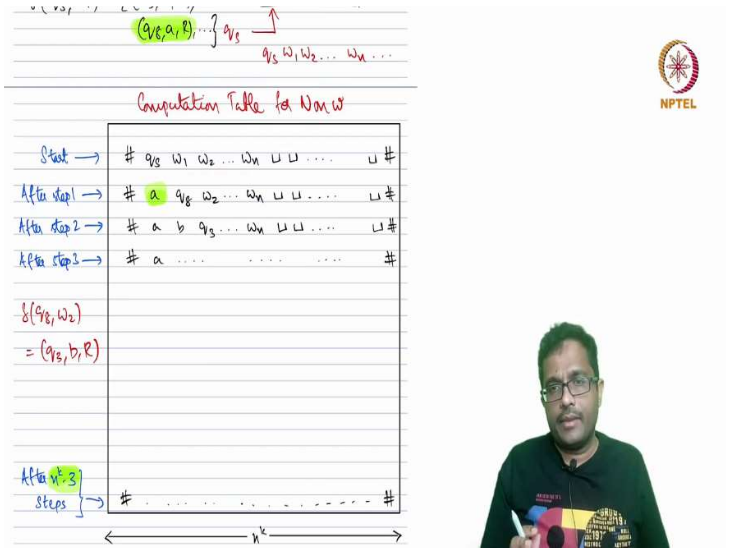
And, so the point is this starting from the starting configuration, if there is a way for w to be accepted by the Turing machine N, the non-deterministic Turing machine N, there is a valid sequence of computations which eventually leads it to accept, so somewhere here accept should be there, somewhere it should accept, the how long it may take to accept.

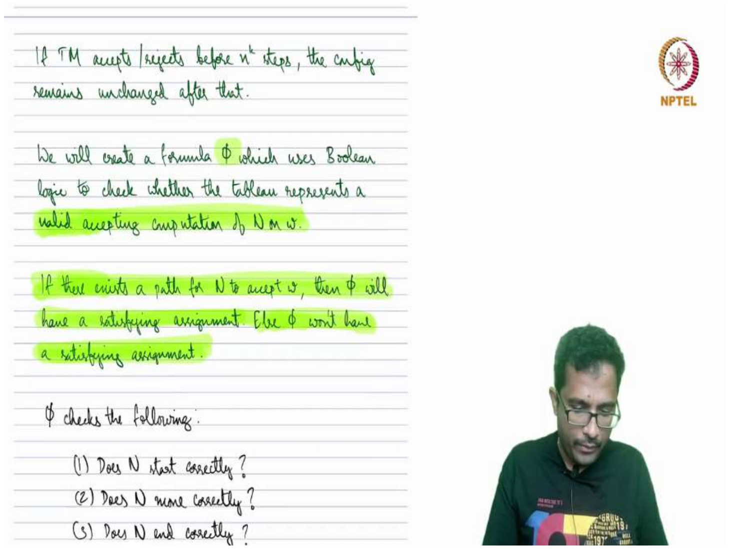
So, we know that it runs in n^k steps, or $n^k - 3$ steps. So, the number of rows of this table need not be anything more than that. So, we know that the acceptance or rejection will happen in n^k steps. So, that many rows is what we have. So, the table has roughly n^k rows and n^k columns.

And if w is accepted by N, there is a sequence of configurations, that starting with the starting configuration, ending with an accepting configuration, where every step is a valid successor, and suppose w is not accepted by the non-deterministic Turing machine N, then there is no such valid sequence of computations that lead it to acceptance.

And this is what we are going to use in order to build the reduction. So, this is a computation table, it has n^k rows, and n^k columns. And basically what we have is the first row indicate the starting configuration, second row indicates after one step, third row indicates after two steps and so on.

(Refer Slide Time: 17:23)


$(q_8, a, R), \dots \} q_s$
$q_s w_1 w_2 \dots w_n \dots$
Computation Table for N on w
Start
$\# \; q_s \; w_1 \; w_2 \dots w_n \sqcup \sqcup \dots \sqcup \#$
After step 1
$\# \; a \; q_8 \; w_2 \dots w_n \sqcup \sqcup \dots \sqcup \#$
After step 2
$\# \; a \; b \; q_3 \dots w_n \sqcup \sqcup \dots \sqcup \#$
After step 3
$\# \; a \dots \dots \dots \#$
$\delta(q_8, w_2) = (q_3, b, R)$
After $n^k - 3$ steps
$\# \dots \#$
n^k



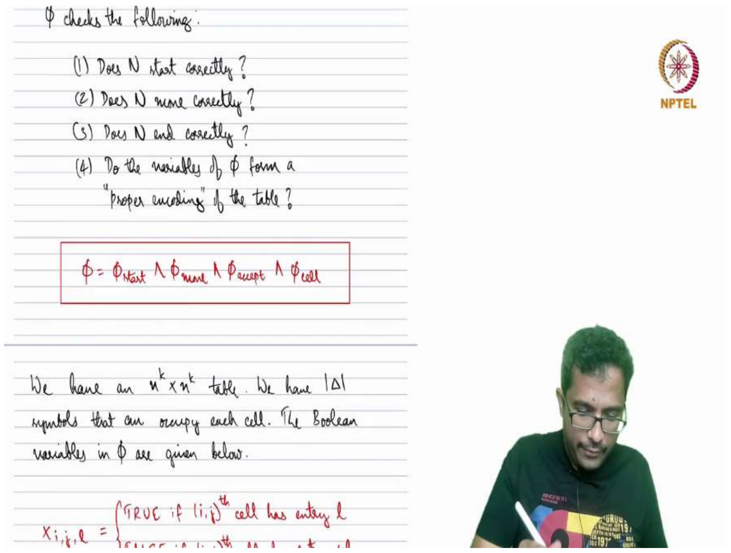
If TM accepts/rejects before n^k steps, the config remains unchanged after that.

We will create a formula ϕ which uses Boolean logic to check whether the tableau represents a valid accepting computation of N on w.

If there exists a path for N to accept w, then ϕ will have a satisfying assignment. Else ϕ won't have a satisfying assignment.

ϕ checks the following:

(1) Does N start correctly?
(2) Does N move correctly?
(3) Does N end correctly?



ϕ checks the following:

(1) Does N start correctly?

(2) Does N move correctly?

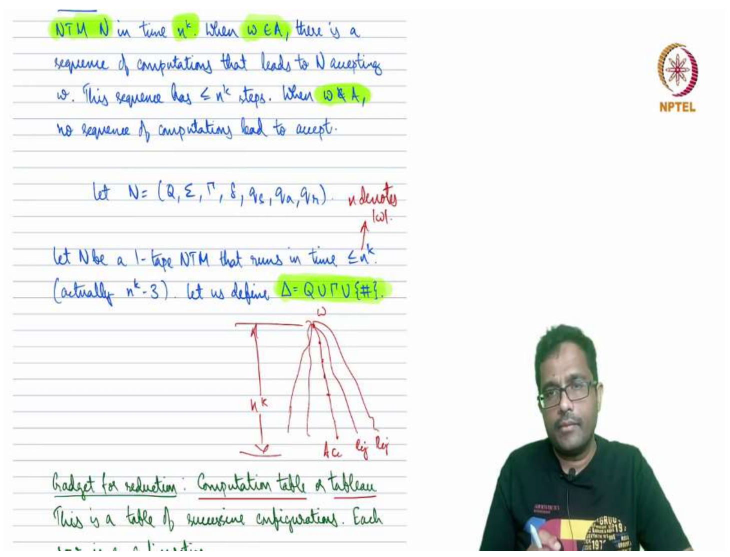
(3) Does N end correctly?

(4) Do the variables of ϕ form a "proper encoding" of the table?

$$\boxed{\phi = \phi_{start} \wedge \phi_{move} \wedge \phi_{accept} \wedge \phi_{cell}}$$

We have an $n^k \times n^k$ table. We have $|\Delta|$ symbols that can occupy each cell. The Boolean variables in ϕ are given below.

$$x_{i,j,l} = \begin{cases} \text{TRUE if } (i,j)^{th} \text{ cell has entry } l \end{cases}$$



NTM N in time n^k. When $w \in A$, there is a sequence of computations that leads to N accepting w. This sequence has $\leq n^k$ steps. When $w \notin A$, no sequence of computations lead to accept.

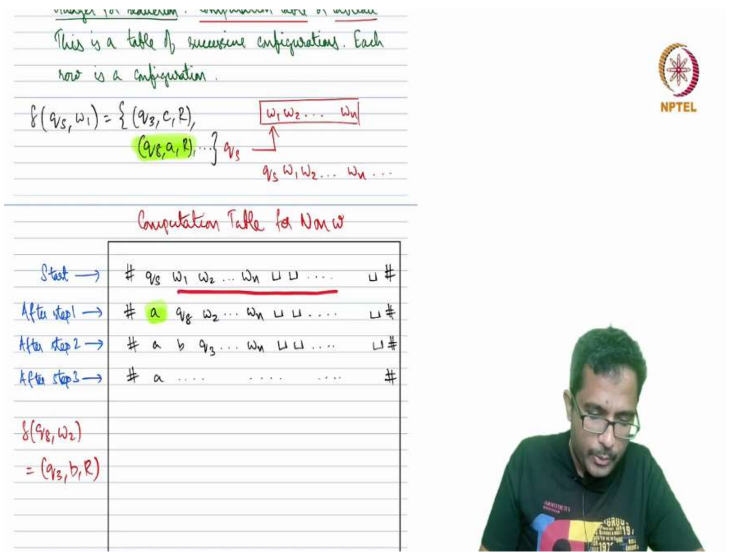
Let $N = (Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$.

Let N be a 1-tape NTM that runs in time $\leq n^k$. (actually $n^k - 3$). Let us define $\Delta = Q \cup \Gamma \cup \{\#\}$.

n denotes $|w|$.

Gadget for reduction: Computation table or tableau

This is a table of successive configurations. Each



This is a table of successive configurations. Each
row is a configuration.

$$\delta(q_s, w_1) = \{(q_3, c, R), (q_8, a, R), \dots\}$$

q_s

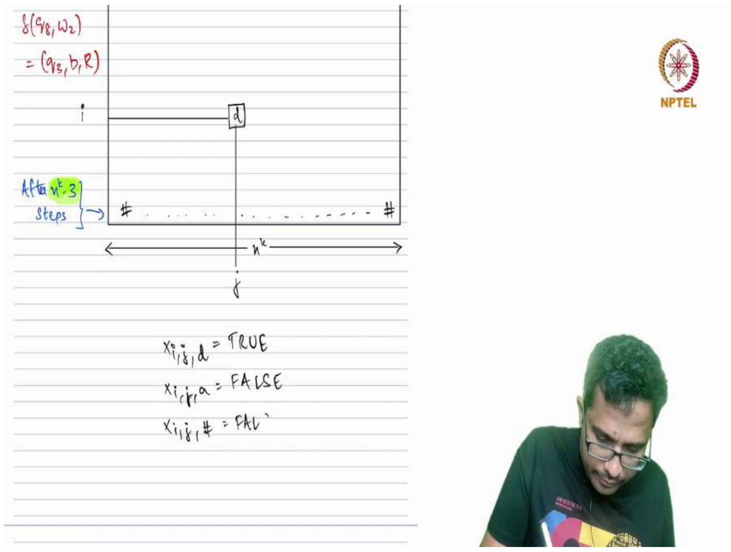
$\boxed{w_1 w_2 \dots w_n}$

$q_s w_1 w_2 \dots w_n \dots$

Computation Table for N on w

Start $\rightarrow$	# q_s w_1 w_2 … w_n ⊔ ⊔ …. ⊔ #
After step1 $\rightarrow$	# a q_8 w_2 … w_n ⊔ ⊔ …. ⊔ #
After step2 $\rightarrow$	# a b q_3 … w_n ⊔ ⊔ …. ⊔ #
After step3 $\rightarrow$	# a …. …. …. #

$\delta(q_8, w_2)$
$= (q_3, b, R)$



And if, the Turing machine accepts or rejects before n^k steps, then we do not change the configuration, it remains unchanged. The configuration, we may assume that, we just repeat the configuration. So that is not really such an important detail. And the formula that we want to create, the formula, we will call it φ, it will check, is there a valid sequence of computations that leads to accepting for the Turing machine N on w.

Basically, whatever I just said, if N accepts w, there is a valid sequence of computations, and the formula is asking is there a valid sequence of computations? The formula will be encoded in such a way that it will have a satisfying assignment if and only if, this computation table has a valid sequence of computations starting from the actual starting configuration for N on w, ending with an accept.

So, the formula will be satisfiable if and only if there is an accepting computation for N on w. If there is no accepting computation for N on w, formula will not have a satisfying assignment. So that is very, very important. Maybe I will just highlight that as well. If there is a path for N to accept w, then formula will have a satisfying assignment, else it will not have a satisfying assignment.

So, we have told the idea, the idea is to somehow encode this computation table and check whether there is a sequence of computations that lead to accept. So, the formula will have to check multiple things, the formula will have to check. To check that this is a valid sequence of computations it will have to check is the first row the valid starting configuration of N on w?

Then it has to check, is the second row a valid successor of the first row, and then it has to check is the third row a valid successor of the second row, and so on, and so on. And finally, it has to check does the computation sequence lead to accept. So, these are the things that we need to check, the formula needs to check, and we will encode all of this kind of in separate modules.

So, the formula φ will have actually four parts, the formula φ will be an and of four things, which we will call phi start, phi move, phi accept, and phi cell.

$$\varphi = \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept} \wedge \varphi_{cell}$$

So, what does each of these do? φ_{start} checks whether the starting configuration is a valid starting configuration of N on w. φ_{move} checks does, is each row a valid successor of the previous row.

So, is the second line a valid successor of the first line, is the third line a valid successor the second line, is the fourth line a valid successor of the third line, and so on and so on. So, all of this together is checked by φ_{move}. φ_{accept} checks that does this table denote an accepting computation?

So, this is what we want to check phi start, phi move, and phi accept. And phi cell is sort of a technical thing where basically we have to capture this entire table into Boolean variables. So, phi cell is basically to ensure that this Boolean variables correspond to a valid encoding of an actual computation table. So, the Boolean variables has to capture this table.

Now, the Boolean variables if it is not set based on some rules it may not lead to a proper, it may not be reflecting a table. So, phi cell is a Boolean formula that indicates that it is actually a table that has been captured by the formula. So, it is a bit of a technical thing. However, it is not that difficult to see. So, once we see. So, when I write down the formula, we will see.

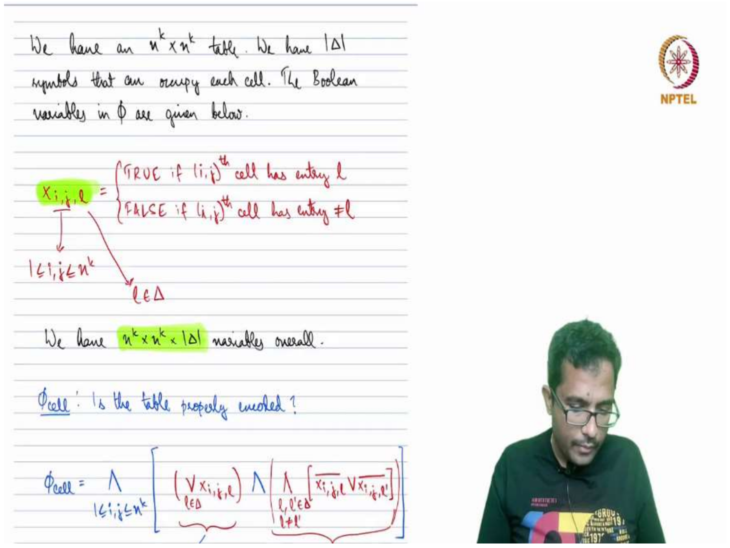
So, this is what we built, we built phi that is able to check whether N has an accepting computation on w. So, phi has four parts, phi start, phi move, and phi accept. Sorry, phi accept, and phi cell, and we will define each of these one by one. So, we have an n^k by n^k table, and we have each table, or each table entry can be either a hash, or a state, or something in the tape alphabet.

So, basically it can be anything from the set that we defined here delta. So, each entry of the table could be an entry from delta. So, for each table entry, we have delta many variables, because each table cell, we can write any of the possible delta symbols. So, each cell can have any of the possible data symbols, and to denote this we have delta sized, or delta variables for each cell.

So, the Boolean variables that we work with are of the form $X_{i,j,l}$ where the, three subscripts i, j, and l. So, i and j denote the table row and column. So, let us say this is i^{th} row, and let us say this is j^{th} column, and let us say the symbol is let us say d. So, what this means is that, if this is the case we want $X_{i,j,d}$ to be true.

So, this denotes that the i^{th} row and the j^{th} column, there is a cell here, this $(i,j)^{th}$ cell contains entry d. So, now, any other thing like $X_{i,j,a}$ will be false, because the variable i, j, a denotes that the $(i,j)^{th}$ cell contains the symbol a, which we know is false because it contains d. So if this is the situation $X_{i,j,d}$ should be true, and $X_{i,j,a}$, or $X_{i,j,b}$ or i, j, anything else let us say $X_{i,j,\#}$ should also be false.

(Refer Slide Time: 25:12)



We have an $n^k \times n^k$ table. We have $|\Delta|$ symbols that can occupy each cell. The Boolean variables in ϕ are given below.

$$\underbrace{\underbrace{x_{i,j},_{l}}_{1\le i,j\le n^k}}_{l\in\Delta} = \begin{cases} \text{TRUE if } (i,j)^{th} \text{ cell has entry } l \\ \text{FALSE if } (i,j)^{th} \text{ cell has entry} \neq l \end{cases}$$

We have $n^k \times n^k \times |\Delta|$ variables overall.

$\underline{\phi_{cell}}$: Is the table properly encoded?

$$\phi_{cell} = \bigwedge_{1\le i,j\le n^k}\left[\underbrace{\left(\bigvee_{l\in\Delta} x_{i,j,l}\right)} \wedge \underbrace{\left(\bigwedge_{\substack{l,l'\in\Delta\\ l\neq l'}}\left[\overline{x_{i,j,l}} \vee \overline{x_{i,j,l'}}\right]\right)}\right]$$

(1) Does N start correctly?

(2) Does N move correctly?

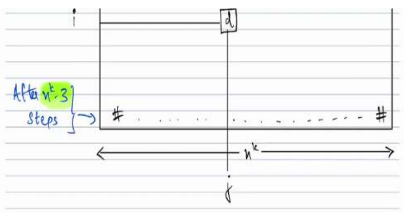
(3) Does N end correctly?

(4) Do the variables of ϕ form a "proper encoding" of the table?

$$\boxed{\phi = \phi_{start} \wedge \phi_{move} \wedge \phi_{accept} \wedge \phi_{cell}}$$

We have an $n^k \times n^k$ table. We have $|\Delta|$ symbols that can occupy each cell. The Boolean variables in ϕ are given below.

$$x_{i,j,l} = \begin{cases} \text{TRUE if } (i,j)^{th} \text{ cell has entry } l \\ \text{FALSE if } (i,j)^{th} \text{ cell has entry} \neq l \end{cases}$$



$x_{i,j,d} = \text{TRUE}$

$x_{i,j,a} = \text{FALSE}$

$x_{i,j,\#} = \text{FALSE}$

~~$x_{i,j,b} = \text{TRUE}$~~

If TM accepts/rejects before n^k steps, the config remains unchanged after that.

$1 \leq i,j \leq n^k$

$l \in \Delta$

We have $n^k \times n^k \times |\Delta|$ variables overall.

$\underline{\phi_{cell}}$: Is the table properly encoded?

$$\phi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} \left[\underbrace{\left(\bigvee_{l \in \Delta} x_{i,j,l} \right)}_{\text{Cell } (i,j) \text{ contains at least one entry}} \wedge \underbrace{\left(\bigwedge_{\substack{l,l' \in \Delta \\ l \neq l'}} \left[\overline{x_{i,j,l}} \vee \overline{x_{i,j,l'}} \right] \right)}_{\text{Cell } (i,j) \text{ does not contain } >1 \text{ entry.}} \right]$$

So ϕ_{cell} ensures that each cell (i,j) contains exactly one member of Δ.

So, this is what $X_{i,j,l}$ denotes. So, i and j denote the row and column, l denotes a member of delta, and this Boolean variable, it is a Boolean variable, it only takes true and false, indicates whether the $(i,j)^{th}$ cell contains the entry l. So, it will be true if the $(i,j)^{th}$ cell contains entry l, and false if the $(i,j)^{th}$ cell contains some other entry, not l.

So, we have i and j range up to 1 to n^k, and l takes Δ possible values. So, we have $n^k * n^k * \Delta$ possible Boolean variables, that many variables is what we have. So, that is what we have. So, now, the first thing that we will define is φ_{cell} , which is a sort of technical thing that I mentioned.

This indicates whether the table is properly encoded, meaning we have a bunch of variables $X_{i,j,d}$, $X_{i,j,a}$, $X_{i,j,\#}$ etcetera, do these encodings denote a proper table? Proper meaning something that is valid. So, thing is this, suppose just for the sake of assumption, assume that $X_{i,j,b}$ was also true. So, we know that $X_{i,j,d}$ is true, and we say that let us say $X_{i,j,b}$ is also true. Now, this means that the cell (i,j) contains both d and b, this cannot happen.

This kind of an absurd thing. So, this is something we do not want to have, we want each cell to contain only one symbol, we cannot have two symbols in a cell.

And second is that, what if all the delta things, X i, j something every one of them is false, which means this cell is empty, that is also something we do not want because if it is empty, the X i, j blank should be true. So, that is why the blank symbol is there. So, it cannot be the case that X i, j everything is false, we need at least one symbol to be there in that spot. And we do not want more than one symbol to be true.

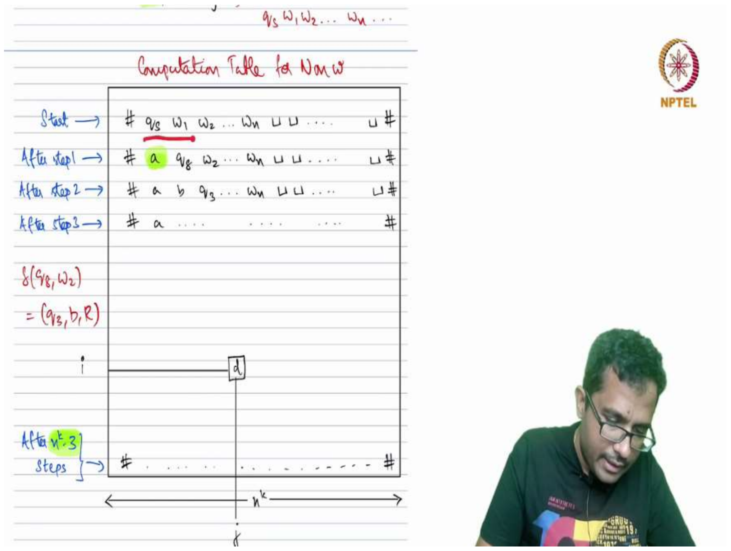
So, this is something we do not want. And this is what basically φ_{cell} checks. So, first, let us focus on the part inside. So, what is this, so, think about. So, now, the outside loop is something where we fix i and j. So, let us look at the part inside. So, this part says that it is an OR across all $X_{i,j,l}$. So, assume that i and j are fixed over all the l. So, now, if we want this particular thing to be true, we want this to be true as well.

So, which means for the cell ij at least one l should be that, at least one symbol should be there. So, which means at least one of these $X_{i,j,l}$ should be true, it cannot be that all $X_{i,j,l}$ is false, because it has to contain something. Second is that, this part is that it cannot be the two symbols are there in the same spot. So, $X_{i,j,l}$ and $X_{i,j,l'}$ both cannot be true. So, we want at most one of them to be true, and that is what is checked by this the, when both of them are true this becomes false.

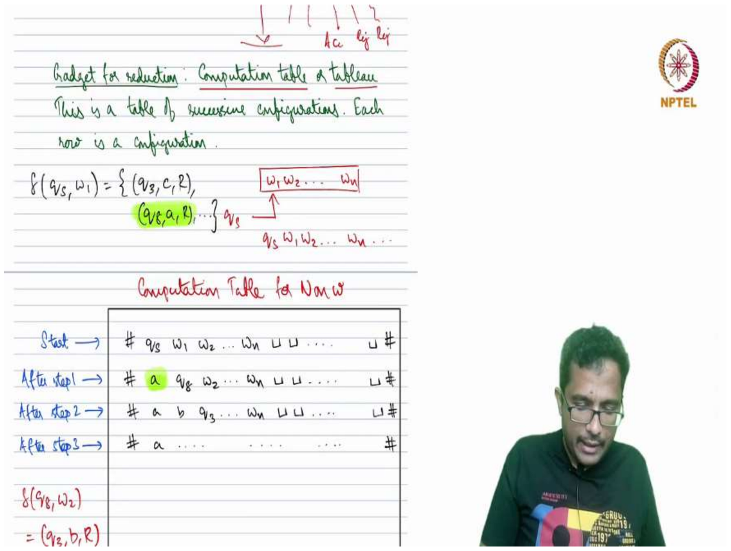
So, this this formula $\overline{X_{i,j,l}}$, or $\overline{X_{i,j,l'}}$ checks that at most one of them is there in the cell i,j, this is what this encodes. And this we want to be verified for all pairs l and l', where l is not the same as l'. So, this part indicates that the $(i,j)^{th}$ cell contains some entry, at least some entry, this indicates that $(i,j)^{th}$ cell does not contain more than one entry.

So, basically this is what I mean by, are the variables properly encoding it in actual table. So, I am not even bothered what the table contains, the table can contain garbage, but is it capturing one entry per each cell of the table that is what the formula φ_{cell} indicates. And outside it, we have an AND over all possible i and j. So for each cell, we want it to contain at least one entry but no more than one entry, that is what this formula is checking. So it ensure that each cell i,j contains exactly one symbol, that is what φ_{cell} indicates.

(Refer Slide Time: 30:11)

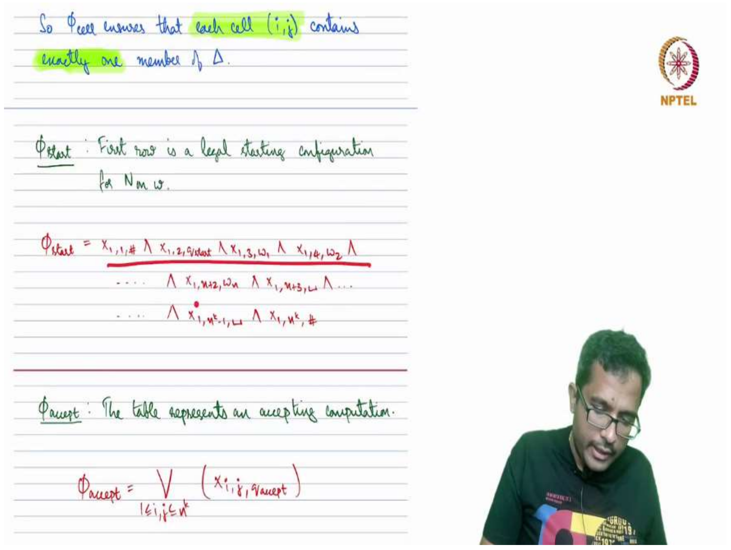


$q_s w_1 w_2 \ldots w_n \ldots$
Computation Table for N on w
Start →
After step 1 →
After step 2 →
After step 3 →
# q_s w_1 w_2 … w_n ⊔ ⊔ … ⊔ #
# a q_8 w_2 … w_n ⊔ ⊔ … ⊔ #
# a b q_3 … w_n ⊔ ⊔ … ⊔ #
# a … #
$\delta(q_8, w_2) = (q_3, b, R)$
i
d
After n^k-3 steps
n^k
j
NPTEL


Gadget for reduction: Computation table or tableau

This is a table of successive configurations. Each row is a configuration.

Acc
l_{ij} l_{ij}
$\delta(q_s, w_1) = \{(q_3, c, R), (q_8, a, R), \ldots\}$ q_s
$w_1 w_2 \ldots w_n$
$q_s w_1 w_2 \ldots w_n \ldots$
Computation Table for N on w
Start →
After step 1 →
After step 2 →
After step 3 →
# q_s w_1 w_2 … w_n ⊔ ⊔ … ⊔ #
# a q_8 w_2 … w_n ⊔ ⊔ … ⊔ #
# a b q_3 … w_n ⊔ ⊔ … ⊔ #
# a … #
$\delta(q_8, w_2) = (q_3, b, R)$
NPTEL



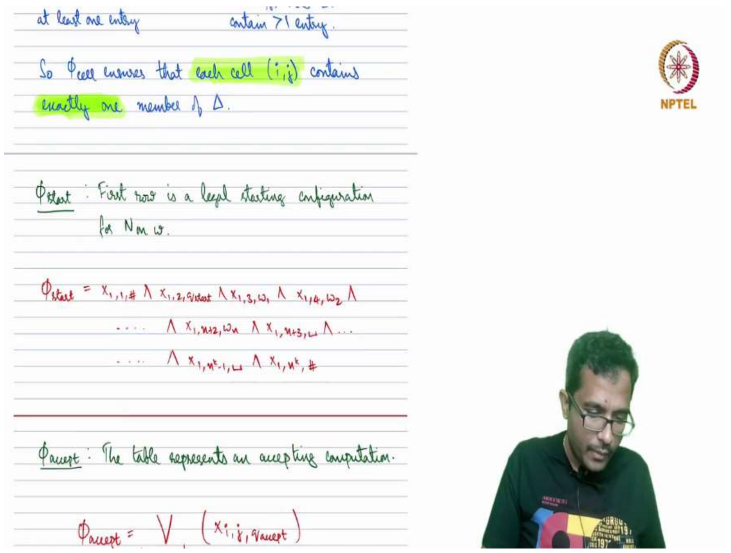
So ϕ_{cell} ensures that each cell (i,j) contains exactly one member of Δ.

$\underline{\phi_{start}}$: First row is a legal starting configuration for N on w.

$$\phi_{start} = \underline{x_{1,1,\#} \wedge x_{1,2,q_{start}} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge}$$
$$\ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \ldots$$
$$\ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

$\underline{\phi_{accept}}$: The table represents an accepting computation.

$$\phi_{accept} = \bigvee_{1 \le i,j \le n^k} \left(x_{i,j,q_{accept}} \right)$$



at least one entry

contain >1 entry.

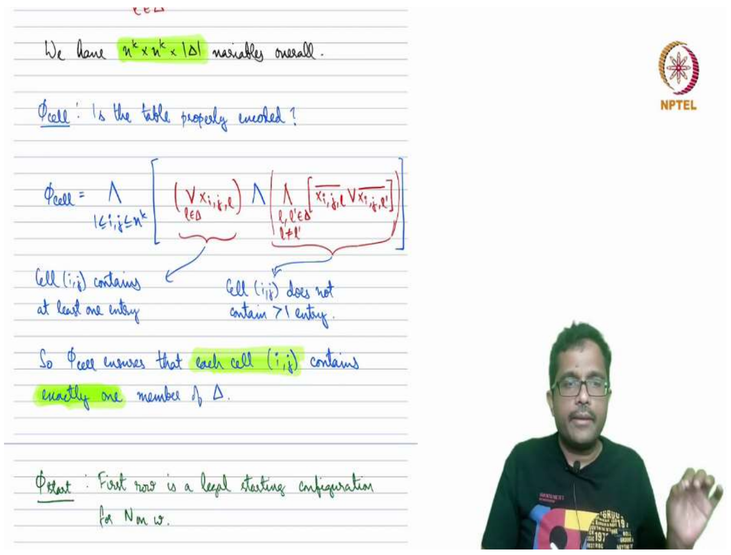
So ϕ_{cell} ensures that each cell (i,j) contains exactly one member of Δ.

$\underline{\phi_{start}}$: First row is a legal starting configuration for N on w.

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_{start}} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge$$
$$\ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \ldots$$
$$\ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

$\underline{\phi_{accept}}$: The table represents an accepting computation.

$$\phi_{accept} = \bigvee \left(x_{i,j,q_{accept}} \right)$$



$l \in \Delta$

We have $n^k \times n^k \times |\Delta|$ variables overall.

$\underline{\phi_{cell}}$: Is the table properly encoded?

$$\phi_{cell} = \bigwedge_{1 \le i,j \le n^k} \left[\underbrace{\left(\bigvee_{l \in \Delta} x_{i,j,l} \right)}_{\text{Cell } (i,j) \text{ contains at least one entry}} \wedge \underbrace{\left(\bigwedge_{\substack{l,l' \in \Delta \\ l \neq l'}} \left[\overline{x_{i,j,l}} \vee \overline{x_{i,j,l'}} \right] \right)}_{\text{Cell } (i,j) \text{ does not contain } >1 \text{ entry.}} \right]$$

So ϕ_{cell} ensures that each cell (i,j) contains exactly one member of Δ.

$\underline{\phi_{start}}$: First row is a legal starting configuration for N on w.

Next is φ_{start}, which is not that difficult. φ_{start} has to check whether the starting configuration is a valid starting configuration of N on w. So, basically, all it has to do is check whether the first symbol is hash, second symbol is the starting state, third symbol is w1, fourth symbol is w2, and so on. And then the is it a bunch of blanks and finally hash, and we can just write a formula directly for this.

So is the first row and first column contain hash, first row and second column contain q_{start}, first row and third column contains w1, fourth column contains w2. So, first row and $(n + 2)$th column contains wn, (n+3)th column contains blank, and so on till the last penultimate column contains blank, and the last column contains hash.

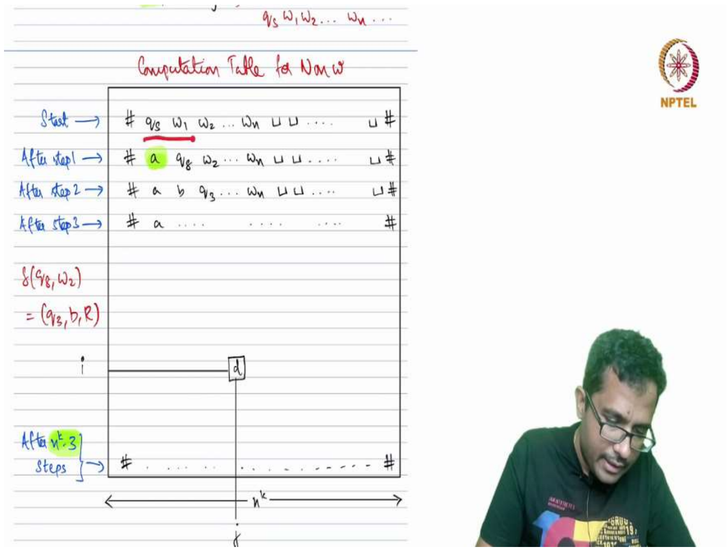
So, basically, we are directly writing down for each column entry, this thing. So, if φ_{start} is true when only when the first row is exactly what is written here, the first symbol has to be hash, the second has to be q_{start}, third has to be the w1, and so on. This is what φ_{start} is encoding, it is just directly encoding the first row of the, first legal starting configuration of N on w.

Now, before going to φ_{move}, because φ_{move} is going to be the most involved thing in this proof, we will also, we will have a look at φ_{accept}. So φ_{accept} indicates that this computation needs to accept. So, we need some accepting state somewhere in this table it could be anywhere we need. So, all we do is we check all the cells is there accepting state somewhere.

So, in all the cells (i, j), we check is there an accepting state somewhere, and that is what we do here, φ_{accept} is we have an OR over all the possible i and j, and we are checking is any of them q_{accept}. So is q_{accept} anywhere in the computation if it is there, that is it. So, we just need anywhere in this table somewhere to have q_{accept}, and that will ensure that φ_{accept} is true.

So, φ_{cell} will be true if the variables encode the table in a proper manner, each cell contains exactly one symbol, φ_{start} will be true if the starting row of the table is the correct starting configuration of N on w, and φ_{accept} will be true if there is a q_{accept} somewhere in the table. So, this is how φ_{start}, φ_{accept} , and φ_{cell} are defined.

(Refer Slide Time: 33:24)



$q_s\, w_1 w_2 \dots w_n \dots$

Computation Table for N on w

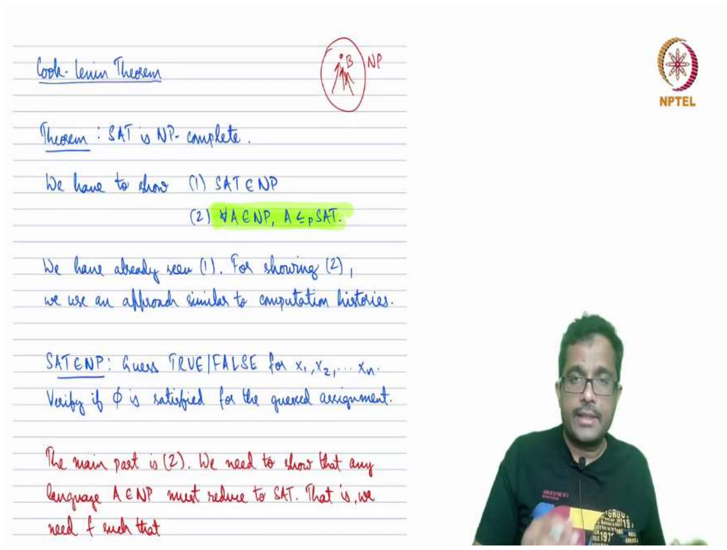
Start $\longrightarrow$	#	q_s	w_1	$w_2 \dots$	w_n	⊔ ⊔	....		⊔	#
After step 1 $\longrightarrow$	#	a	q_8	$w_2 \dots$	w_n	⊔ ⊔	....		⊔	#
After step 2 $\longrightarrow$	#	a	b	$q_3 \dots$	w_n	⊔ ⊔	....		⊔	#
After step 3 $\longrightarrow$	#	a	....		....		....			#
After n^k-3 steps $\longrightarrow$	#	.	....	..	....		.....			#

$\delta(q_8, w_2)$

$= (q_3, b, R)$

i $\boxed{d}$ j

$\longleftarrow n^k \longrightarrow$



Cook-Levin Theorem

NP B

Theorem: SAT is NP-complete.

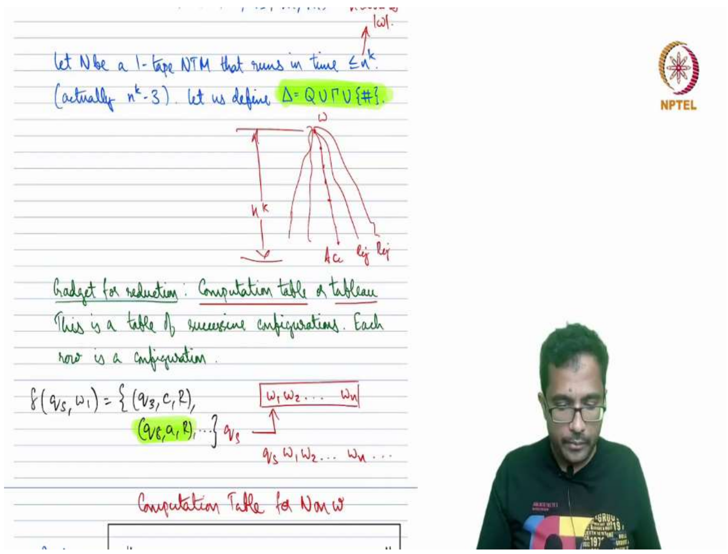
We have to show (1) SAT $\in$ NP

(2) $\forall A \in NP,\ A \leq_p SAT$.

We have already seen (1). For showing (2), we use an approach similar to computation histories.

SAT $\in$ NP: Guess TRUE/FALSE for $x_1, x_2, \dots x_n$. Verify if ϕ is satisfied for the guessed assignment.

The main part is (2). We need to show that any language $A \in NP$ must reduce to SAT. That is, we need f such that



$|w|$.

Let N be a 1-tape NTM that runs in time $\leq n^k$. (actually n^k-3). Let us define $\Delta = Q \cup \Gamma \cup \{\#\}$.

Gadget for reduction: Computation table or tableau

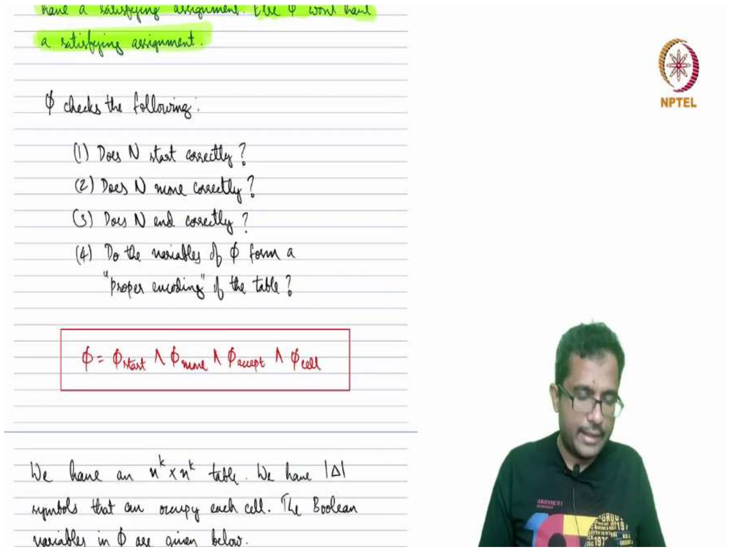
This is a table of successive configurations. Each row is a configuration.

$\delta(q_s, w_1) = \{(q_3, c, R),$
$(q_8, a, R), \dots\}$ q_s

$\boxed{w_1 w_2 \dots w_n}$

$q_s\, w_1 w_2 \dots w_n \dots$

Computation Table for N on w


have a satisfying assignment. Else ϕ won't have a satisfying assignment.

ϕ checks the following:

(1) Does N start correctly?
(2) Does N move correctly?
(3) Does N end correctly?
(4) Do the variables of ϕ form a "proper encoding" of the table?

$\boxed{\phi = \phi_{start} \wedge \phi_{move} \wedge \phi_{accept} \wedge \phi_{cell}}$

We have an $n^k \times n^k$ table. We have $|\Delta|$ symbols that can occupy each cell. The Boolean variables in ϕ are given below.

And then the next thing that is to be shown is φ_{move}. So, here φ_{move} has to capture that each row is a valid successor of the previous row. So, second row is a valid successor of first row, third row is valid successor of second row, and so on. And this will be slightly more involved. And since this is kind of a lengthy proof, I thought we will break at this point, and resume the rest of the proof in the next lecture.

So, just to summarise, what we saw here to show the SAT is NP-complete, we first showed that SAT is in NP, then we wanted to reduce an arbitrary NP language A to SAT, for that we formed this computation table. And the goal is that if a string w is accepted by the NTM decider for A, there is a valid sequence of computations that lead for it to be accepted.

And those valid sequence of computations will make the formula have a satisfying assignment. So whatever formula we are building, if there is no valid sequence of computations that lead to accept, the formula will not have a satisfying assignment. So that is the thought process, and we said that the formula will have four parts, that capture whether the machine starts correctly, moves correctly, ends correctly and whether it is encoded properly.

We saw three of them, we saw the starting, ending and the encoding part, but the part that checks whether each move is a proper move, we will see in the next lecture. So, that is all from me in lecture number 50. But to see the rest of the proof of Cook-Levin theorem, please watch lecture number 51. Thank you.