


Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Cook-Levin Theorem - Proof and Implications

(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 to show that

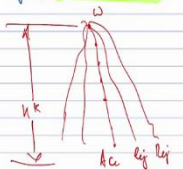






$A \in$ NP: This means that A is decided by an NTM N in time n^k . When $w \in A$, there is a sequence of computations that leads to N accepting w . This sequence has $\leq n^k$ steps. When $w \notin A$, no sequence of computations lead to accept.

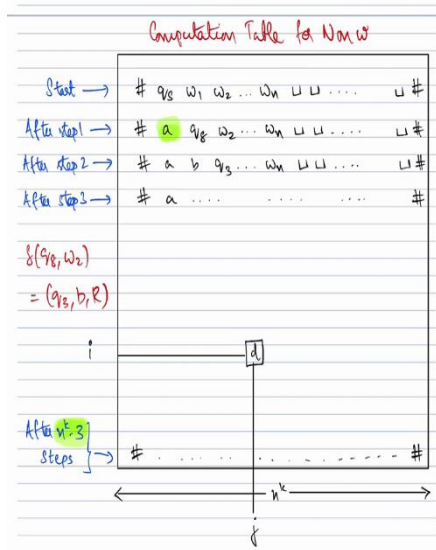
Let $N = (Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$. n denotes $|w|$.

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









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

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



Hello and welcome to lecture 51 of the course theory of computation. In lecture 50, we started the proof of Cook-Levin theorem. So, the Cook-Levin theorem states that SAT is NP-complete. So, which involves showing that SAT is in NP, and all the languages in NP can be reduced to SAT. So, the first part is easy, the second part is where the challenge of the proof lies, and the reduction goes.

So, we have to take an arbitrary NP language A , and we have to show that A reduces to SAT. So, the way we went about it is using this computational table. So, we showed that the idea was to construct a Boolean formula, which has a satisfying assignment if and only if the given string w is in A .

So, in other words, instead of checking, directly checking w is a , we want to check whether the decider for a , the non-deterministic decider N for a , does there exists a valid computation for N on w , that will lead to w getting accepted on N . So, this is what we want the formula to check. So, let me repeat, we want the Boolean formula ϕ to encode whether there exists an accepting computation for N on the string w .

So, n is a non-deterministic decider for the language A . So, w is in A if and only if the non-deterministic decider can accept w , and that can happen only if there is a sequence of configurations that leads into acceptance, and this is what we want to encode in the Boolean formula ϕ . And the details we discussed in the previous lecture. So, we had to check four things.

One is whether the formula, whether the configurations is the starting with a valid starting configuration, and does it end with an exit, and whether the variables encode the table properly. So, these three things, these three parts we saw in the last lecture, the part that was remaining was to check whether each configuration listed here is that a valid successor of the previous configuration. So, this is the part that is remaining to explain. So, this part of the formula is called ϕ_{move} , and this is what we have to discuss.

(Refer Slide Time: 3:04)

how is a configuration ..

$$\delta(q_s, w_i) = \{ (q_s, c, R), (q_s, a, R), \dots \}$$

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

w_1, w_2, \dots, w_n


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_s	w_2	\dots	w_n	\sqcup	\sqcup	\dots	\sqcup	#
After step 2 \rightarrow	#	a	b	q_s	\dots	w_n	\sqcup	\sqcup	\dots	\sqcup	#
After step 3 \rightarrow	#	a	\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	#

$\delta(q_s, w_2)$
 $= (q_s, b, R)$

i d

After $n-3$







Φ_{accept} : The table represents an accepting computation.

$$\Phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq N} (X_i, i, \Phi_{\text{accept}})$$

Φ_{move} : This is the hardest. We have to check that each configuration legally follows from the previous one.

The main idea here is that it is enough to check all the 2×3 windows. We say a window is valid if it can be part of a valid transition.

$b \ q \ a$ $n \ b \ d$ $n \ b \ d$ $n \ b \ d$

$b \ q \ a$ When $(n, d, R) \in \delta(q, a)$
 $b \ d \ n$ $\forall b \in \Gamma \cup \{\#\}$

$b \ q \ a$ When $(n, d, L) \in \delta(q, a)$
 $n \ b \ d$ $\forall b \in \Gamma$

$\# \ q \ a$ When $(n, d, L) \in \delta(q, a)$
 $\# \ q \ d$

$b \ c \ q$ If $(n, d, R) \in \delta(q, a)$
 $b \ c \ d$ for some $a \in \Gamma$, where
 $b \in \Gamma \cup \{\#\}$, $c \in \Gamma$
Not in the window

$q \ a \ b$ If $(n, d, R) \in \delta(q, a)$
 $d \ n \ b$ where $b \in \Gamma \cup \{\#\}$



We can list all the valid windows in this manner. The number of valid windows is finite and depends only on N . It is independent of $|W|$.

Claim: If all the 2×3 windows are valid, then all C_i are legally followed by C_{i+1} .

Proof: (1) If a symbol is not adjacent to the state, then it is unchanged.

Since because middle symbol is unchanged in windows that do not have state.

(2) If a symbol is adjacent to state, it appears in a window where the state is in the top middle.

This window captures a valid move for N .



Claim: If all the 2×3 windows are valid, then all C_i are legally followed by C_{i+1} .

Proof: (1) If a symbol is not adjacent to the state, then it is unchanged.

Hence because middle symbol is unchanged in windows that do not have state, in the top row of the window.

(2) If a symbol is adjacent to state, it appears in a window whose state is in the top middle.

This window captures a valid rule for N .



So, the point is that we have to check for each the second configuration is valid successor of the first, the third is valid successor of the second, and so on. And the main idea here is that the Turing machine computation is a very local phenomenon, and instead of actually checking the entire configurations, we will do a sort of a local check.

So, what we do is we will divide the entire, think of there being like windows like this. So, when I say window, I am talking about a part of the table which has two rows and three columns. So, what we will you do is think about this window moving, so one step. So, now, we check this window, and the next time we check the window, similar window but move one step to the right.

And then we check another window that moves one more step to the right, and after which you check some window like this. And like one step below and so on. So, basically the tile the entire table using these windows. So, of course, there will be overlaps and we want the overlap to be there. So, for all possible such 2 by 3 windows, we want to check some things.

So, what do we want to check? I will explain that. So, basically we define or we divide the entire table, actually divide is not the right word, cover is the right word, we cover the entire table using these 2 by 3 windows, and there will be overlap. So, if there is a window like, there will be maybe I will use another colour, the next window will be something like this.

So, when everything move one step forward, and the next window will be again this, and yet another window will look something like this. So one step below, so, we cover the entire table using these 2 by 3 windows and then we check these 2 by 3 windows are valid, this is going to

be the strategy. Think about the entire table covered with this 2 by 3 windows, and we are going to check all these windows if they are valid.

So now, what constitute valid windows is what we have to explain, and why covering and checking these 2 by 3 windows is enough? So, we come to the formula. So we want to check each configuration whether it is a valid successor of the previous one, and the idea is that it is enough to check all the 2 by 3 windows. What is a valid window? A window is a valid window if it is a part of a valid transition.

So if there are two configurations, and for instance, let us say we saw here, the window that is marked over here. The a, q_8, w_2, a, b, q_3 . So, the q_8 was in the centre here, and after some valid move w_2 was overwritten by b . So, this is a valid window because it appears.

And so, will this be, this will also be a valid window after whatever appears here, so here maybe w_3, w_3, q_8, w_2, w_3 and below b, q_3, w_3 , that is also a valid window. Anything that can appear as part of a valid computation is a valid window. So, now let us just to get a feel for this let us try to understand. So, suppose as part of a computation, you have b, q, a . Meaning, the way to look at it is if you look at the Turing machine, so you have b here and you have a here, and the head is pointing at a , and with state q .

So what we doing is that suppose this tells us to move right one step. And a is overwritten by some symbol let us say d . And then maybe something else is read, and q becomes r , the next state is r . So, that is captured by this thing here, when we are reading q_a one of the possible moves is to write d onto the tape, and then go to state r , and then move one step right. So, that is what this indicates $\delta(q, a)$ contains r, d, r .

So, notice it is a non-deterministic Turing machine. So, we do not say $\delta(q, a)$ equal to, this is one of the many possible moves, or one of the some possible moves that we do not know how many, and where b could be any symbol. So, when this happens, the first row had b, d, b, q, a and the next after one step, the b remains as it is the d , the a becomes d , but then it moves one step to the left, because the tape had moves one step to the right, and you have r .

And whatever is the tape head pointing it will be here, but then that is not part of the 2 by 3 window. So, that will get captured in the window when we look at in the next, when we move it to the next step. So, this is a valid window, b, q, a, b, d, r . So, in fact, the thing that we saw

here was similar, we have a, q, a, w_2 , and a, b, q_3 . So the roles of a's and b's have changed around, but you see why this is a valid window.

Another thing is that same the same b, q, a, but instead of writing b going to state r and moving right, we move left. So, the only difference here is that r, d, l is in q_a . So the in that case a gets replaced by d. Now, the tape head moves one step left, so q becomes r, and it moves one step left and b comes here. So this is a valid window as well. Now, this is the case when b is a symbol, and then the tape head can move one step to the left.

What if this was a hash symbol? That is considered in the next case. So, which means this hash symbol is there only in the leftmost end. So now the tape head cannot go left. So in that case, it remains in the same place. So again, the same rule r, d, l is in q_a but if it is, if the adjacent left symbol is hash, it cannot move left, because you are already at the left most SAT.

So these are some of the valid computation, 2 by 3 windows, there could be even more. So for instance, so all these three situations had q in the centre. What if q is in the right? What if it is reading some symbol a over here? What if it was reading some symbol a over here? And then it moves right, writes d and goes to state r. So then in the right here, it would be. So the a, b gets replaced by d, and q goes to r here.

But then this is not part of the window, the window is just the part that is highlighted in blue. This is not part of the window. So, b, c, q, b, c, d is also a valid window, if you have this kind of a rule, for any A, because A is not part of this window, we do not, it does not matter for which a, and whichever b, whichever c and so. In fact, we can also be hashed here.

And so another case when, there will be a similar case when if this the tape head moves left, it will be different. But again, I am not enumerating all the possible windows, I am only telling, giving some ideas so that you get a flavour of what are the windows possible, there are many types of windows possible, maybe another 7, 8 types are there, but if I just keep going through the list of windows, it will become very, very, the lecture could just become quite monotonous.

So what if the q was in the top right, sorry, top left, now, if the there is a tape, so now the tape it is reading a and it has to replace it with d and move one step right, the one in the bottom is d, r, b. So, again is the same rule r, d, r is in q, a. Now, another pause. So another possibility is so here we all have here. So these three considered q in the centre, top centre, here q was in the top left, or top right, here q was in the top left.

Again, I did not considering the case when q moves left from here, or the tape head moves left from here, I am only considering the case when tape head moves right. Once again, the goal is to not enumerate all the possible list of windows, but just give do some so that you get an idea of what all is possible.

Finally, another thing that is possible is some a, b, c at the top and the same a, b, c in the bottom, maybe the state is somewhere far away. So these things do not change. So there is also a valid window. Here also, we had said such valid windows here? So let us say w_2, w_3, w_4 it is the same thing in the bottom also, any window that is over here will not change because the ones that will change is where the tape head is.

So, which is indicated by the state in the configuration. So, the same thing in the top and bottom also is a valid window. So, these are some example of valid windows. The point is, the valid window is a function of only the Turing machine. So given, once the machine is fixed, the set of all valid windows is also fixed. It depends on only the Turing machine, the number of states, the number of variables, sorry, the number of tape alphabet, the transition rules, etc.

It has no connection with the input, what is input? So, the number of valid windows is a constant. Constant, meaning it is independent of the input string, but it is dependent on the Turing machine. So that is what I have written here. The number of valid windows is dependent only on the Turing machine N , and is independent of the input string.

In particular, this is independent of the length of the input string. So now, the point is that it is enough to check all these 2 by 3 windows? So we will try, or we will cover the entire table using these 2 by 3 windows. So all these cells will get covered by multiple windows, because there is overlap, we want all the overlap. Only then can we ensure that things are checked thoroughly. And if you.

So, the point that if we verify that all the 2 by 3 windows are valid, then it means that each configuration is a valid successor of the previous configuration. Or in other words, each configuration is legally followed by the next configuration. It is enough to check, this claim says that it is enough to check all the 2 by 3 windows. So why is that? So suppose a symbol is not adjacent to the state. Suppose the symbol is not adjacent to the state.

So maybe I will just write down here. Suppose a symbol let us say it is a here is somewhere and the state is somewhere far. So, now, the point is that there is a 2 by 3 window, which does

not contain the state at all, if a is not adjacent to the state, which means it is either side is not the state, so, we can consider the 2 by 3 window with a in the top centre, and because it is not adjacent to the state, so the worst case is that the state could be here, but it could be anywhere, but it will not be adjacent to the a.

So, now, the thing is that the state could be here to the left of c, or to the, sorry to the right of c, or to the left of b, but whatever it is now, the symbol a will be unchanged in the next row. Because, the q may enter here in the bottom left, it may also enter in the bottom right, q or the next state may enter, but it will not change a. So, all the.

So, if you go through the list of all windows, all the valid windows, so, you do not have to go through you can just think about it for yourself, the top middle symbol, so, this is a window where the top row does not have the state, this is a window where the top row does not have the state, and in such windows the middle symbol in the top row is unchanged is the same in the bottom row. So, this is what I want to say.

So, this a does not change so, the if the symbol is not adjacent to the state it is unchanged, this is what we want. So, there is one I just write it here this is true because the middle symbol is unchanged in the windows that do not have the state maybe in the top row of the window, or not of course, it stayed in the top row somewhere but not in the window.

So, if the symbol is not adjacent to the state, then it is, then the that symbol is unchanged which is true in the configuration as well, and this is reflected by the window in which that symbol is in the top middle. Second, suppose the symbol is adjacent to the state, we want to say that it will change in the in a proper way as per the rules of the Turing machine, that is the second thing.

So, if a symbol is adjacent to the state, we want to say it gets modified only according to the rules of the Turing machine. Why is that? Suppose a is adjacent to the state. So, maybe it is so a here and q here and let us say b here. Now, the thing is that then there is a 2 by 3 window, which features the state in the top middle. So a is adjacent to the state, then there is a window where the state is in the centre.

So maybe a, q, c, and now the claim is that this window captures completely what happens to the because of the transition, because q is in the top middle. Now that q may move right or left, or q will become let us say it moves left, then c becomes d or something and a comes here. So

now the movement is only according to the, so this particular 2 by 3 window in which the state is in the top centre, completely captures what happens to the to this part of the tape, the changes are entirely confined to this 2 by 3 window.

So, if a symbol is adjacent to the state, that symbol will feature in the window where the state is in the top middle. And that window captures completely what happens, what changes happen. So, that window will capture everything and so, that is also fine. So, if the symbol is not adjacent to the state, then it is unchanged.

And if a symbol is adjacent to the state, it features in this kind of window, where the state is in the top mid, and that captures all the changes. Hence, whatever changes happen are only near the state and that is captured, and whatever is not near the state, it is unchanged because of the first part. So that completes the proof of the claim.

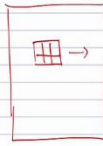
So, anything that is away from the state is left untouched, and anything that is near the state is captured by the window in which the move is captured correctly. Hence, this ensures that any configuration is a valid successor of the previous configuration, and also there is overlap, there is sufficient overlap which ensures things are reflected properly. So, this is why we need to tile the thing with overlap, because we want every symbol that is adjacent to state, we want it to look at the windows where the state is in the top middle.

So, now, that this is done, so, now, we have shown that the number of we have told what are the possible windows, we have given some examples, this is not an exhaustive list, but the list of valid windows is a finite number which only depends on the Turing machine, and not on the input. And second, it is enough this claim says that it is enough to check all the 2 by 3 windows.

(Refer Slide Time: 20:46)

$$\bigvee \begin{matrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \end{matrix} \left[x_{1,i+1,a_1} \wedge x_{1,i,a_2} \wedge x_{1,i+1,a_3} \wedge \right. \\ \left. x_{1+1,i-1,a_4} \wedge x_{1+1,i,a_5} \wedge x_{1+1,i+1,a_6} \right]$$

is a valid window



We have shown that

$$\phi \in \text{SAT} \Leftrightarrow N \text{ accepts } w \\ \Leftrightarrow w \in A$$

We also need to show that the reduction, i.e. ϕ , can be constructed in poly time.

$$\text{There are } n^k + n^k + |\Delta| \text{ variables} = O(n^{2k})$$

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

Cell: Is the table properly checked?

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,j \leq n^k} \left[\underbrace{\left(\bigvee_{\ell \in \Delta} x_{i,j,\ell} \right)}_{\text{Cell (i,j) contains at least one entry}} \wedge \underbrace{\left(\bigwedge_{\ell_1 \neq \ell_2 \in \Delta} \neg (x_{i,j,\ell_1} \wedge x_{i,j,\ell_2}) \right)}_{\text{Cell (i,j) does not contain >1 entry.}} \right]$$

Cell (i,j) contains at least one entry

Cell (i,j) does not contain >1 entry.

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

ϕ_{start} : First row is a legal starting configuration for N on w .

$$\phi_{\text{start}} = v \wedge \dots$$

ψ_{start} : First row is a legal starting configuration for N on w .

$$\phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_{\text{start}}} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \wedge x_{1,n^k,w_{n^k-1}} \wedge x_{1,n^k,\#}$$

ϕ_{accept} : The table represents an accepting computation.

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,j \leq n^k} (x_{i,j,q_{\text{accept}}})$$

ϕ_{move} : This is the hardest. We have to check that each configuration legally follows from the previous one.



Now we can write Φ_{move} .

$i-1$	i	$i+1$
$j-1$	j	$j+1$

$$\Phi_{move} = \bigwedge_{1 \leq i \leq n^k} \left(\text{the } (i,j)\text{th window is legal} \right)$$

a_1	a_2	a_3
a_4	a_5	a_6

is a valid window

x_{i-1, a_1}	\wedge	x_{i, a_2}	\wedge	x_{i+1, a_3}	\wedge
$x_{i, j-1, a_4}$	\wedge	x_{i, j, a_5}	\wedge	$x_{i, j+1, a_6}$	\wedge

<table style="border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 2px;">+</td> <td style="border: 1px solid black; padding: 2px;">+</td> </tr> </table>	+	+	→
+	+		

We have shown that

$$\Phi \in \text{SAT} \Leftrightarrow N \text{ accepts } w$$

$$\Phi \in \text{SAT} \Leftrightarrow N \text{ accepts } w$$

$$\Leftrightarrow w \in A$$

We also need to show that the reduction, i.e. Φ , can be constructed in poly time.

There are $n^k \times n^k \times |A|$ windows = $O(n^{2k})$

↓
independent of $|w|$.

$\Phi_{cell} = O(n^{2k})$ length

$\Phi_{start} = n^k$ length

$\Phi_{accept} = n^{2k}$ length

$\Phi_{move} = \text{No. of valid windows} \times n^{2k} \times 6$

$= O(n^{2k})$



So, now we can write down the Φ_{move} . So, what is Φ_{move} ? We go through all possible windows i, j is the. So, the i, j th window what I mean is by the i, j th window what I mean is this. So, this is i and this is j so, this will be $i, j + 1$, this will be $i + 1, j$, this will be $i + 1, j + 1$ and so on. So, here will be $i, j - 1$.

So, maybe I just make it a bit bigger. This is i, j there is $i + 1, j, i, j + 1, i + 1, j + 1, i + 1, j - 1$ and $i, j - 1$. So, when you say i, j th window, I mean the window where i, j is in the top middle. So, we want to check all the i, j th, all the windows there they are legal. So what I mean is, so, we look at all the windows and check whether this window is a legal window by exhaustively going through the list of valid windows.

So, this part the part that is so we check whether it is legal, how do we check it is legal. So we know the variables here? What we do is for each term like this is expanded like here. So where we take in OR of the all the valid windows, and we check whether this window corresponds to that valid window. So if $a_1, a_2, a_3, a_4, a_5, a_6$ is a valid window, we are checking whether the $i, j - 1$ entry is a 1, i, j entry a 2, $i, j + 1$ that is a 3 and so on till $i + 1, j + 1$ entry is a 6.

So it is an AND over all the windows, and all over all the valid windows and inside there is and AND, whether this window is this valid window. So whether the selected window i, j corresponds to the specific valid window, $a_1, a_2, a_3, a_4, a_5, a_6$, so like that, we check through all the valid windows, so maybe it is not this valid window, but maybe it is some other valid window.

So, we maybe there are 1000 valid window. So we need to have this all that runs over all the 1000 valid windows. Maybe there are 10,000. Maybe there is 1 million valid windows, but it is a constant that is independent of the length of the input, it will not change if the input is longer or smaller. So, this is ϕ_{move} , you go through all the windows i, j and check whether.

So, when I say Windows i, j I mean the entries i, j in the table and check whether this entry is equal to a valid window by just going through all the list of valid windows, and checking, is this where is this basically we are doing this maybe this small pictorial explanation might help. So, we have a 2 by 3 window that is moving across the table. And once we fix a moving, once we fix a 2 by 3 window, we check whether, there is a list of valid windows given. So, valid window 1 to 1000.

So, is this the first valid window, no, is the second window, no, is it third valid window, and so on. And at some point it is, if it is a valid window that part will say yes. And that will make this class yes. So that is what we are doing here. So, we are going through lists of all valid windows and checking whether this particular window chosen is this does this equal any specific valid window. So that completes the reduction. That completes the construction.

So first let us see the correctness. So, correctness has been established by what we said in the previous lecture and this lecture. That if we are checking whether this so this formula is satisfiable if and only if we can set these variables in a certain way, and we can set these variables in a certain way only when there is a sequence of computations that lead N to accepting w.

So, this formula is satisfiable if and only if N accepts w , if and only if that happens if and only if w is in A , because N is a decider for A , even though a non-deterministic decide. Now, so, this is the correctness, what remains is show the running time of the construction. So, the running time of the construction is also not that difficult, because if you see all the all the part, all the formula even though it was the for instance this is like n^k times n^k etcetera, and here we are running a loop on delta, here again double loop on delta.

However, the fact that I can write the formula in a concise manner like this indicates that you could I can write a loop, or some or loop or multiple loops, nested loops or something for creating this formula. So, for instance, ϕ_{cell} I can have two loops outside that run that where i and j run from 1 to n^k , and one loop that runs through all symbols in delta here, and then one double loop that runs through like l runs through all symbols of delta, and l' runs through all symbols of delta, and output this formula.

Similarly, if I start is basically, I have X_1 , let us say j and a specific symbol. So, this formula also, this also can be encoded in a loop. So, very easily very, very clear structure kind of formula. So, which can easily be written down in the program. And ϕ_{accept} , you just have $X_{i,j,q}$ accept where i and j vary, all vary from 1 to n^k . So, this is also straightforward. ϕ_{move} is also straightforward.

So, we have an outside AND that runs through all i and j , and within that there is an OR that runs through all the valid windows, and then inside that there is a ϕ , it is AND of 6 letters. So, if you look at the length of each of these formula, the number of variables which I had, I think mentioned in the previous lecture also there is $X_{i,j,1}$ where it ranges from 1 to n^k , j ranges from 1 to n^k , and l ranges from has size delta.

So, the number of variables is n^k times n^k times delta, but delta is independent of the length of w . So, it is, it gets absorbed in O . So, it is $O(n^{2k})$, because delta gets absorbed in the O notation.

(Refer Slide Time: 27:57)

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

$1 \leq i, j \leq n^k$ $l \in \Delta$

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

Φ_{cell} : Is the table properly checked?

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

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

$1 \leq i, j \leq n^k$ $l, l' \in \Delta, l \neq l'$

Cell (i,j) contains at least one entry Cell (i,j) does not contain > 1 entry.

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

Φ_{start} : First row is a legal starting configuration for N_{hw}.

$$\Phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_{\text{start}}} \wedge x_{1,3,q_1} \wedge x_{1,4,q_2} \wedge \dots \wedge x_{1,n^k-1,q_n} \wedge x_{1,n^k,\#}$$

Φ_{acc} : The table represents an accepting computation.

$$\Phi_{\text{acc}} = x_{1,1,\#} \wedge x_{1,2,q_{\text{start}}} \wedge x_{1,3,q_1} \wedge x_{1,4,q_2} \wedge \dots \wedge x_{1,n^k-1,q_n} \wedge x_{1,n^k,\#}$$

Φ_{accept} : The table represents an accepting computation.

$$\Phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} (x_{i,j,q_{\text{accept}}})$$

Φ_{move} : This is the hardest. We have to check that each configuration legally follows from the previous one.

The main idea here is that it is enough to check all the 2×3 windows. We say a window is valid



Now we can write ϕ_{move} .

$$\phi_{move} = \bigwedge_{1 \leq i, j \leq n} \text{the } (i, j) \text{th window is legal}$$

is a valid window

$$\left[\begin{array}{l} x_{i, i-1, a_1} \wedge x_{i, i, a_2} \wedge x_{i, i+1, a_3} \wedge \\ x_{i+1, i-1, a_1} \wedge x_{i+1, i, a_2} \wedge x_{i+1, i+1, a_3} \end{array} \right]$$


can be constructed in poly time.

There are $n^k \times n^k \times |\Delta|$ variables = $O(n^{2k})$

independent of $|\Delta|$.

$\phi_{cell} = O(n^{2k})$ length

$\phi_{start} = n^k$ length

$\phi_{accept} = n^{2k}$ length

$\phi_{move} = \left. \begin{array}{l} \text{No. of valid} \\ \text{windows} \end{array} \right\} \times n^{2k} \times b$

$= O(n^{2k})$

So ϕ is poly size in n . There is an easy algorithm to generate ϕ , as it has a repetitive structure.



Now, ϕ_{cell} if you see this is n^{2k} , but inside it is just dependent on the delta. So, here there is delta, here there is a square, so, it is still $O(n^{2k})$, because this part is constant once i and j are fixed. ϕ_{start} basically there is only one row. So, it has only n^k not even O , this is just n^k variables. And ϕ_{accept} has actually n^{2k} variables, again not even asymptotic exactly n^k variables.

That is what I have written here ϕ_{cell} has $O(n^{2k})$ length. ϕ_{start} has n^k length, ϕ_{accept} has n^{2k} length. So, there is not even O here. ϕ_{move} for instance has $O(n^{2k})$ length because of this. So, this has n^{2k} , and checking with the i, j th windows legal is this runs through the list of all valid windows and this is constant.

So, number of valid windows in constant, number of valid windows is also a constant that is independent of the length. So, number of valid windows into n^{2k} into 6, this also n^{2k} . So the size of the formula is polynomial in n, everything is n^{2k} , n^k , n^{2k} etcetera.

And as we noted they are all very, very concise descriptions, which means that there is a way to we can easily write with some for loops an algorithm to generate this formula, because it has a very, very repetitive structure. So, that completes the proof that SAT is NP-complete.

(Refer Slide Time: 29:41)

n^k

↓ ↓ ↓ ↓ ↓

k c q_i q_j q_k

Budget for reduction: Computation table of tableau

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

$f(q_s, w_i) = \{ (q_s, c, r), (q_s, a, b), \dots \}$

$w_1 w_2 \dots w_n$

$q_s w_1 w_2 \dots w_n \dots$

Computation Table for $W_n w$

Start →	#	q_s	w_1	w_2	...	w_n	⊔	⊔	...	⊔	#
After step 1 →	#	a	q_s	w_2	...	w_n	⊔	⊔	...	⊔	#
After step 2 →	#	a	b	q_s	...	w_n	⊔	⊔	...	⊔	#
After step 3 →	#	a	#

$f(q_s, w_2)$

↓

$\phi_{accept} = n^{2k}$ length

$\phi_{none} = \left. \begin{matrix} \text{No. of valid} \\ \text{windows} \end{matrix} \right\} \times n^{2k} \times b$


$= O(n^{2k})$


So ϕ is poly size in n. There is an easy algorithm to generate ϕ , as it has a repetitive structure.


Theorem: 3-SAT is NP-complete.


One way to show is to show $SAT \leq_p 3-SAT$.

Slightly easier: To modify the above proof so as to generate a 3-CNF SAT instance.









So, the reduction part. So, we took an arbitrary language A that is in NP. So, we assumed that there is a non-deterministic Turing machine N that decides A. So, non-deterministic Turing

machine that runs in n^k time, that decides A. And we encoded the formula that we have created in such a way that the formula accepts, or formula has a satisfying instance.

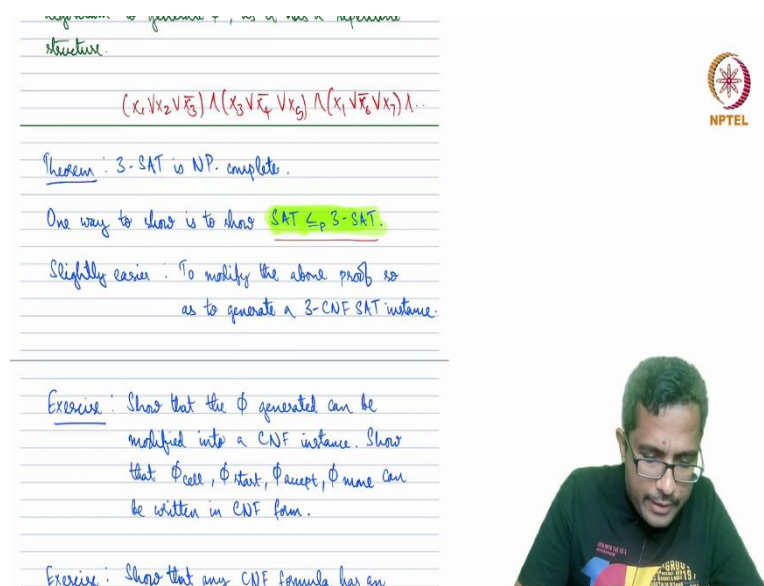
Sorry, formula has a satisfying assignment if and only if there is a sequence of computations that lead N to accept a distinct w. So, N accepts w if and only if the formula is satisfying instance, and that gives the reduction. And the running time is also polynomial. So, that completes is a proof of the Cook-Levin theorem.

And one key thing in the in the proof is that, we noticed instead of checking whether each configuration is a valid successor of the previous one, we see the solid is enough to check the validity of all the 2 by 3 windows, we can basically cover the entire table with a 2 by 3 windows and check the validity of them.

And that helped bring down the size of the formula. So that completes the proof of Cook-Levin theorem. It is a very interesting theorem, but we need to start somewhere. So now that we have the Cook-Levin theorem, now, it should be easy to prove the proof some other languages are NP-complete.

For instance, if we want to show that a language click is NP-complete, we just need to show a reduction from SAT to click, and we need to show that click is an NP. If you want to show that subset sum is NP-complete, we need to show a reduction from SAT to subset sum and show that subset sum is in NP. So we do not need to do everything from scratch where we take an arbitrary language in NP like we took here.

(Refer Slide Time: 31:47)



structure.

$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_4 \vee x_5) \wedge (x_1 \vee \bar{x}_6 \vee x_7) \wedge \dots$$


Theorem: 3-SAT is NP-complete.


One way to show is to show $SAT \leq_p 3-SAT$.

Slightly easier: To modify the above proof so as to generate a 3-CNF SAT instance.

Exercise: Show that the ϕ generated can be modified into a CNF instance. Show that ϕ_{cell} , ϕ_{start} , ϕ_{accept} , ϕ_{time} can be written in CNF form.

Exercise: Show that any CNF formula has an





Now we can write ϕ_{row} .



$$\phi_{\text{row}} = \bigwedge_{1 \leq i \leq n^k} (i,i)^{\text{th}} \text{ window is legal}$$

$$\bigvee \begin{bmatrix} x_{i,i-1,a_1} \wedge x_{i,i,a_2} \wedge x_{i,i+1,a_3} \wedge \\ x_{i+1,i-1,a_4} \wedge x_{i+1,i,a_5} \wedge x_{i+1,i+1,a_6} \end{bmatrix}$$

is a valid window

is a valid window



We have shown that

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?

$$\phi = \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{end}} \wedge \phi_{\text{cell}}$$

We have an $n^k \times n^k$ table. We have $|\Delta|$ symbols that can occupy each cell. The Boolean

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

$1 \leq i, j \leq n^k$
 $l \in \Delta$

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

ϕ_{cell} : Is the table properly encoded?

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

cell (i,j) contains at least one entry cell (i,j) does not contain >1 entry.

So ϕ_{cell} ensures that each cell (i,j) contains



Φ_{start} : First row is a legal starting configuration for N on w .

$$\Phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_{start}} \wedge x_{1,3,q_1} \wedge x_{1,4,q_2} \wedge \dots \wedge x_{1,n+2,q_n} \wedge x_{1,n+3,\#} \wedge \dots \wedge x_{1,n+1,\#} \wedge x_{1,n+2,\#}$$

Φ_{accept} : The table represents an accepting computation.

$$\Phi_{accept} = \bigvee_{1 \leq i, j \leq n} (x_{i,j, q_{accept}})$$

Φ_{move} : This is the hardest. We have to check that



Now we can write Φ_{move} .

$i-1$	i	$i+1$	$right$
$i-1, q_{i-1}$	i, q_i	$i+1, q_{i+1}$	$i+1, q_{i+1}$

$$\Phi_{move} = \bigwedge_{1 \leq i, j \leq n} (i, j)^{th} \text{ window is legal}$$

$$\bigvee [x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}]$$

is a valid window

a_1	a_2	a_3
a_4	a_5	a_6

--	--	--	--

→

We have shown that

$$\Phi \in SAT \Leftrightarrow N \text{ accepts } w$$


So I will conclude this lecture with a couple more things. First thing is that 3-SAT is NP-complete, so what is 3-SAT, 3-SAT is a set of all Boolean formulas that is that are in 3-CNF form, and are satisfiable. So 3-CNF means things like this, something like X_1 . It is an AND of X_1 OR X_2 OR $\overline{X_3}$ and X_3 OR $\overline{X_4}$ OR X_5 and X_1 OR $\overline{X_6}$ OR X_7 something like this.

So it is an AND of ORs, where every clause contains exactly three literals, and every clause is an OR of exactly three literals, so, this is NP-complete. One way is to reduce. So, we have already shown that SAT is NP-complete. So, to show the 3-SAT is NP-complete, one way is to reduce SAT to 3-SAT in polynomial time.

So, in all of this, I am assuming that showing 3-SATs is in NP is straightforward. In fact, we have already discussed it in one of the previous lectures. So, one way to show is that to reduce

SAT to 3-SAT, but one way to show is to do this. Another way instead of making a fresh reduction, what we can do is. So, whatever we proved we did for the Cook-Levin theorem, we can kind of tweak the proof.

So that the, so we got a formula ϕ at the end of the proof. Now, if this formula ϕ is a 3-SAT formula, with an equivalent 3-SAT formula, meaning this formula ϕ satisfies the condition that whenever N accepts w , this ϕ is satisfiable and whenever and does not accept w this ϕ is not satisfiable, and this formula is 3-SAT, or 3-CNF SAT, then this reduction itself will do.

So, it turns out that we can convert this the formula that is output from the Cook-Levin proof into a 3-SAT, 3-CNF SAT instance. So I just list down the way to do it as to exercises, I will not write down and go into detail. So first, so we have the formula is in AND of four things, ϕ_{start} , ϕ_{move} , ϕ_{accept} , and ϕ_{cell} . So CNF means AND of ORs, AND of clauses. So we will show that each one of them is a CNF, and this is not that difficult.

So ϕ_{cell} is basically already an AND of something here. And so again, this can be considered as one clause and this can be considered as another clause. So ϕ_{cell} is already in the form of AND of ORs, so it is okay. ϕ_{start} it is already in the form of AND of individual variables. So we can think of these as AND of singleton clauses, clauses with singleton literals. ϕ_{accept} is basically an OR of many things.

So we can think of this ϕ_{accept} as a single clause, there is no end. Now, ϕ_{move} , it is actually it is an AND of OR of ANDs. So, that is AND of OR of ANDs here, what we can do is, so this, so anything that. So, OR an AND kind of, we can use, we can distribute over each other. So, this OR of ANDs, we can write it as AND of ORs, therefore getting AND of AND of ORs, which again, will become a CNF instance. So here, we need to use the distributive property of OR of, OR over ANDs.

(Refer Slide Time: 35:43)



$$(x_1 \vee x_2 \vee x_3 \vee x_4) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee x_4)$$

$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee x_5)$$

Exercise: Show that 2-SAT ∈ P.

Distributing OR over AND

$$(a \wedge b) \vee (c \wedge d) \Leftrightarrow (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$$



Slightly easier: To modify the above proof so as to generate a 3-CNF SAT instance.



Exercise: Show that the ϕ generated can be modified into a CNF instance. Show that ϕ_{cell} , ϕ_{start} , ϕ_{accept} , ϕ_{name} can be written in CNF form.

Exercise: Show that any CNF formula has an equivalent 3-CNF expression.

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee x_4)$$

$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee x_5)$$



Exercise: Show that any CNF formula has an equivalent 3-CNF expression.



$$x_1 \Leftrightarrow (x_1 \vee x_1 \vee x_1)$$

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee x_4)$$

$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee x_5)$$

Exercise: Show that 2-SAT ∈ P.

Distributing OR over AND's.

$$(a \wedge b) \vee (c \wedge d) \Leftrightarrow (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$$



So for instance, maybe just to give a small example. So we want to show that OR over ANDs, OR of ANDs can be written as AND of ORs. So OR of ANDs, so suppose a and b, OR c and d, this is equivalent, we can write it as a OR c, AND a OR d, AND b, OR c, AND b, OR d. So you can verify that the formula in the left, maybe I will use a different colour, the formula in the left.

This is true, if and only if the formula on the right is true. So this is the left formulas true is if either a and b, either a and b is true, or c and d is true. And here also the same thing is required. So this can be verified. So maybe just so distributing OR over ANDs. So, this way we can convert each of the ϕ_{cell} , ϕ_{start} , ϕ_{accept} , and ϕ_{move} to a CNF instance. And the second thing is at once. So that is the first exercise.

The second thing is that once we get something in CNF form, we can convert them to 3-CNF form. So that is a way so the formula generated, we first ensure that it is in CNF, then we ensure that it is in 3-CNF. So any CNF formula can be converted to 3-CNF. It is not that difficult. So suppose we have a clause. So basically, we modify the clauses to ensure that it has exactly three literals. So suppose we have a clause X_1, X_2, X_3, X_4 .

We can, convert it into AND of two clauses, where we introduce a new variable y. And now you can verify that y has to be either true or false. So if y is true, then X_3 or X_4 has to be true. If y is false, then X_1 , or X_2 has to be true. So basically, by two choices of y, we are covering the entire possibilities of the OR.

And similarly, when we have a clause of 5 literals. We can do the similar thing by introducing two new variables Y_1 and Y_2 , which connect all these 5 original variables. And if there is a clause that contains only 1 variable or something. So if you have something like this, let us say X_1 , you can map it to X_1 OR X_1 OR X_1 . That is also, and next if there is two also you can repeat one of them. So that is another exercise.

(Refer Slide Time: 39:01)

$$(\exists z \vee x_p \vee x_s)$$

Exercise: Show that 2-SAT is P.

2-SAT: CNF formula $C_1 \wedge C_2 \wedge C_3 \wedge \dots$

where each clause is of the form $(x_i \vee x_j)$

Key idea: Every 2-SAT clause can be viewed as an implication.

Distributing OR over AND's.

$$(a \wedge b) \vee (c \wedge d) \Leftrightarrow (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$$



Look: Levin Theorem



Theorem: SAT is NP-complete.

We have to show (1) SAT is NP

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

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

SAT is 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 $A \in \text{NP}$ must reduce to SAT. That is, we



$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge$$

$$(y_1 \vee x_3 \vee y_2) \wedge$$

$$(y_2 \vee x_4 \vee x_5)$$

Exercise: Show that 2-SAT is P.

2-SAT: CNF formula $C_1 \wedge C_2 \wedge C_3 \wedge \dots$

where each clause is of the form $(x_i \vee x_j)$

Key idea: Every 2-SAT clause can be viewed as an implication.

Distributing OR over AND's.

$$(a \wedge b) \vee (c \wedge d) \Leftrightarrow (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$$



$\phi_{\text{true}} = \text{no. of valid windows} \approx n^2 \approx 6$
 $= O(n^2)$



So ϕ is poly size in n . There is an easy algorithm to generate ϕ , as it has a repetitive structure.

$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_4 \vee x_5) \wedge (x_1 \vee \bar{x}_6 \vee x_7) \wedge \dots$

Theorem: 3-SAT is NP-complete.

One way to show is to show $\text{SAT} \leq_p \text{3-SAT}$.

Slightly easier: To modify the above proof so as to generate a 3-CNF SAT instance.



Exercise: Show that the ϕ generated can be modified into a CNF instance. Show that $\phi_{\text{cell}}, \phi_{\text{start}}, \phi_{\text{accept}}, \phi_{\text{true}}$ can be written in CNF form.

Exercise: Show that any CNF formula has an equivalent 3-CNF expression.

$x_1 \Leftrightarrow (x_1 \vee x_1 \vee x_1)$
 $(x_1 \vee x_2 \vee x_3 \vee x_4) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee x_4)$
 $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \Leftrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee x_5)$

Exercise: Show that 2-SAT $\in P$.

2-SAT: CNF formula $C_1 \wedge C_2 \wedge C_3 \wedge \dots$

And finally one more thing that I wanted to mention is that 2-SAT is in P. So we said that 3-SAT is NP-complete, which means it is unlikely that 3-SAT has a polynomial time algorithm, however 2-SAT, so what is 2-SAT? It is 2-CNF formula, which is AND of clauses where each clause is of 2 literals. So 2-SAT CNF formula C_1 AND C_2 AND C_3 AND so on.

Where each clause is of the form something, let us say X_i OR X_j , or something X_j, \bar{X}_a or something like this. It is an OR of two literals So this is surprisingly in polynomial time. So this is something interesting because 2-SAT is in polynomial time while 3-SAT is NP-complete, which we expect to be not polynomial time.

So what happens when we go from 2 to 3, or what happens when we go from 3 to 2. So the reason is, we can view 2-SAT as something like implications. So this, let us say something, we

have X_i , maybe just for the sake of simplicity, I will just take $X_i \text{ OR } X_j$. So it is like, if X_i is false, then X_j has to be true, otherwise, this will not be satisfied. So this can be viewed as implications, and that characterisation helps us show that 2-SAT is in P.

So I will not get into the proof here. But the key idea is that every clause can be viewed as an implication, maybe I will write the idea down. Key idea every 2-SAT clause can be viewed as an implication. So that is how we will show that 2-SAT is in P. That is all, that is pretty much all that I had in lecture number 51, where we completed the proof of Cook-Levin theorem.

So Cook-Levin theorem, we want to show that SAT is NP-complete, we showed by taking arbitrary NP language A, we considered the non-deterministic polynomial time decider for A, and constructed a formula, Boolean formula such that the non-deterministic decider has an accepting computation for a given string, if and only if the given the constructed Boolean formula was satisfiable. And that involves multiple steps in multiple parts, in multiple reasonings, and that completes lecture 51.

This is also the end of week 10, week 10 lectures. In week 10 lecture, we first saw the verifier characterisation of NP, then, we saw a verifier characterisation of NP, then we saw that NP has is the same as polynomial time verifiers, then we saw reductions, we saw a polynomial time reductions, we saw NP-completeness, and then we saw the proof that SAT is NP-complete.

So SAT is NP-complete and using this now, we can now other languages to show that it is NP-complete, we do not need to show that an arbitrary NP language reduces to it. Instead, we can take SAT and reduce the to the other language. So in week 11, we will see and pick up other NP-complete languages where we will not be showing it from scratch, but from some already known NP-complete languages, maybe like SAT or something that we have already proven at that time. So that completes week 10, and see you in week 11. Thank you.