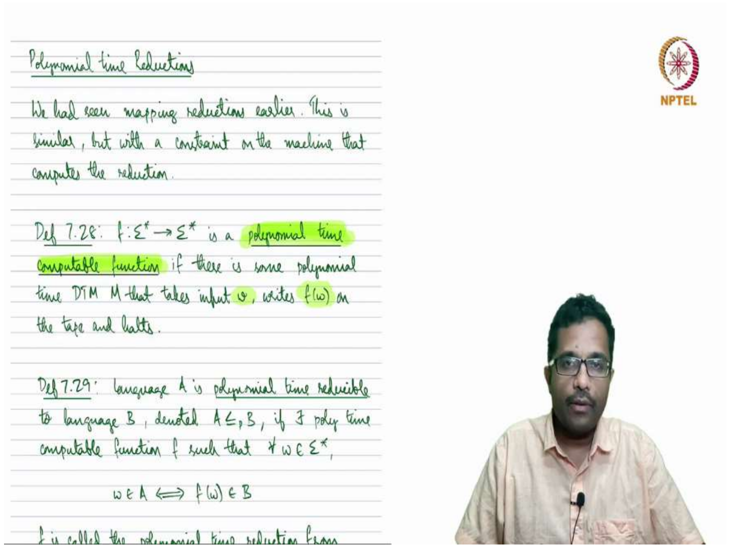
Theory of Computation

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

Polynomial Time Reductions - Part 1

(Refer Slide Time: 00:15)



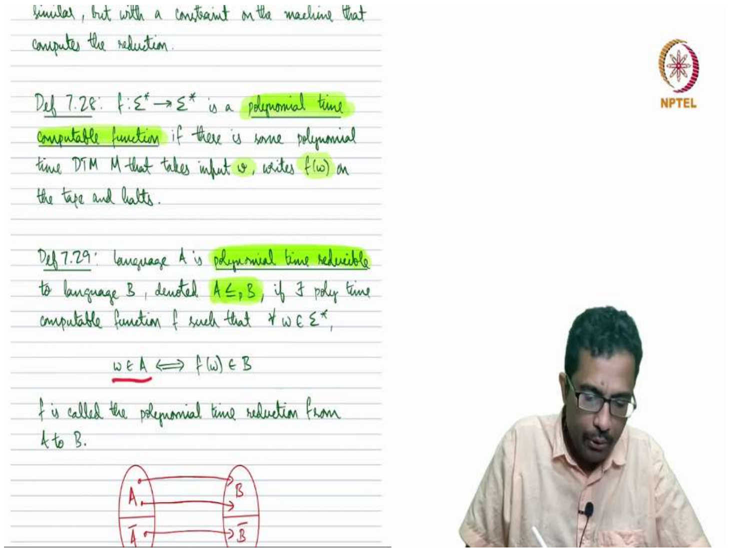
Polynomial time Reductions

We had seen mapping reductions earlier. This is similar, but with a constraint on the machine that computes the reduction.

Def 7.28: $f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if there is some polynomial time DTM M that takes input w, writes $f(w)$ on the tape and halts.

Def 7.29: language A is polynomial time reducible to language B, denoted $A \leq_p B$, if $\exists$ poly time computable function f such that $\forall w \in \Sigma^*$,

$$w \in A \iff f(w) \in B$$



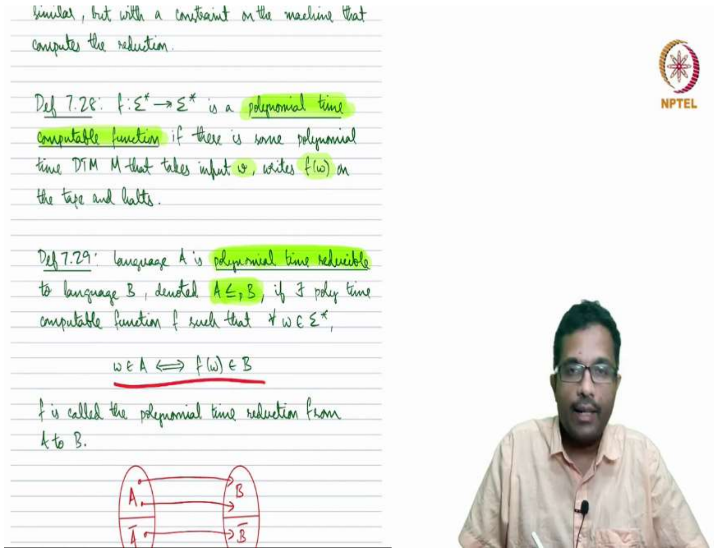
similar, but with a constraint on the machine that computes the reduction.

Def 7.28: $f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if there is some polynomial time DTM M that takes input w, writes $f(w)$ on the tape and halts.

Def 7.29: language A is polynomial time reducible to language B, denoted $A \leq_p B$, if $\exists$ poly time computable function f such that $\forall w \in \Sigma^*$,

$$\underline{w \in A} \iff f(w) \in B$$

f is called the polynomial time reduction from A to B.



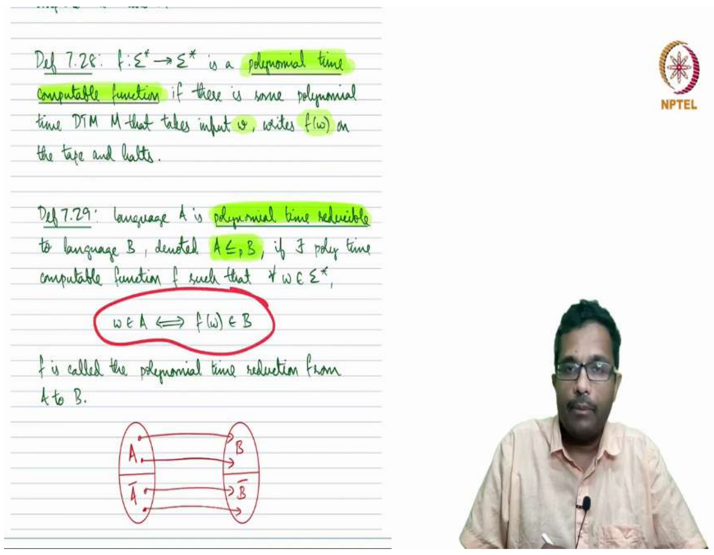
similar, but with a constraint on the machine that computes the reduction.

Def 7.28: $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if there is some polynomial time DTM M that takes input w, writes $f(w)$ on the tape and halts.

Def 7.29: language A is polynomial time reducible to language B, denoted $A \le_p B$, if $\exists$ poly time computable function f such that $\forall\, w \in \Sigma^*$,

$$w \in A \iff f(w) \in B$$

f is called the polynomial time reduction from A to B.

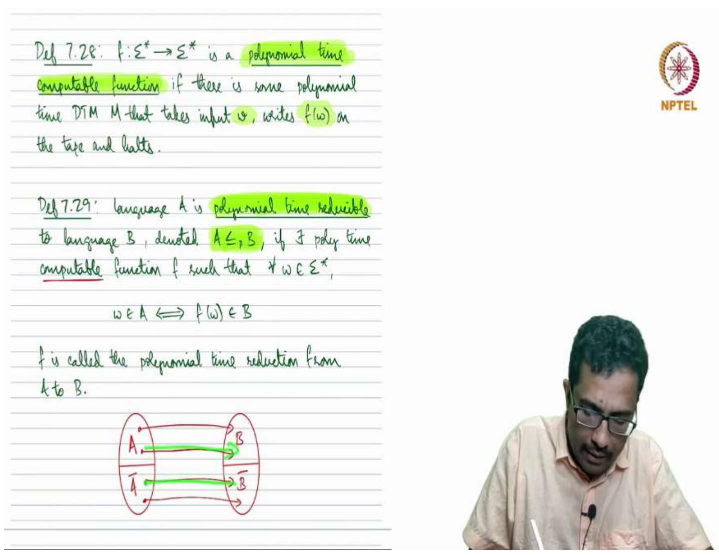


Def 7.28: $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if there is some polynomial time DTM M that takes input w, writes $f(w)$ on the tape and halts.

Def 7.29: language A is polynomial time reducible to language B, denoted $A \le_p B$, if $\exists$ poly time computable function f such that $\forall\, w \in \Sigma^*$,

$$\boxed{w \in A \iff f(w) \in B}$$

f is called the polynomial time reduction from A to B.



Def 7.28: $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if there is some polynomial time DTM M that takes input w, writes $f(w)$ on the tape and halts.

Def 7.29: language A is polynomial time reducible to language B, denoted $A \le_p B$, if $\exists$ poly time computable function f such that $\forall\, w \in \Sigma^*$,

$$w \in A \iff f(w) \in B$$

f is called the polynomial time reduction from A to B.

Hello and welcome to lecture 48 of the course Theory of Computation. In the previous lecture, we saw the verifier characterization of NP. In this lecture, we will see polynomial time reductions. And this will later be helpful towards establishing the notion of NP completeness. So, let us let us get into the definition of polynomial time reductions.

So, we have already seen reductions earlier which was in chapter 5 mapping reductions or many one reductions. So, this is similar, but with a different constraint on the machine computes a reduction. So, what is a reduction? So, first, I am going to define polynomial time computable function.

So, we say that a function f is polynomial time computable if there is a deterministic Turing machine M that that takes the input and provides output. Meaning that function f is polynomial time computable if there is a polynomial time deterministic tuning machine M that takes the input let say w and outputs f(w) on the tape and then stops.

So, there should be a machine that takes w as input and outputs f(w), and this should be done in polynomial time by a deterministic Turing machine. So, this is the definition of polynomial time computable function and we say that a language A is polynomial time reducible to language B and denoted by this notation $A \leq_p B$

If there is a polynomial time computable function f that that constitutes a reduction. What do you mean by reduction? The reduction is that for all w in A, $f(w)$ should be in B and for all w not in A, $f(w)$ should not be in B.

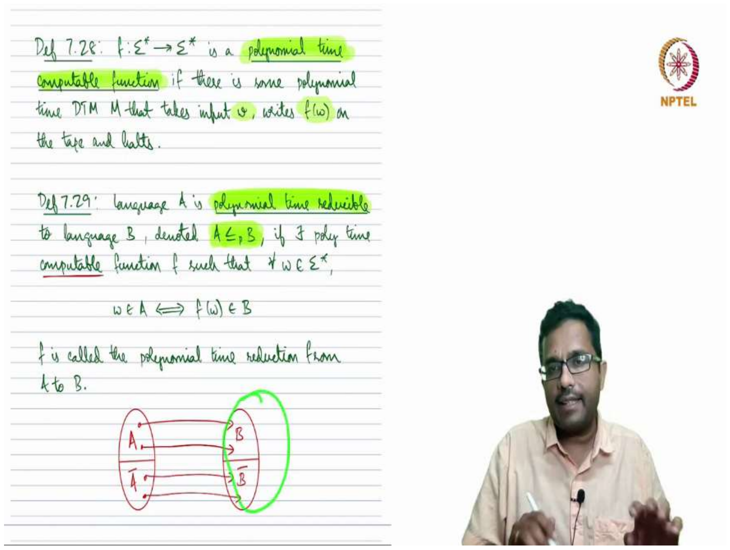
$$w \in A \iff f(w) \in B$$

So, in short w is in A if and only if $f(w)$ is in B. So, this is very similar to the notion of mapping reductions or many one reductions that we saw in chapter 5.

The only difference is that so, even there we had this w is in A then $f(w)$ is in B. The difference is the only difference that here we are insisting that the function that constitutes the reduction the function f should be polynomial time computable. So, this is what I am highlighting this function should be polynomial time computable in the case of a polynomial time reduction, whereas, in the case of mapping reduction, we just want it to be computable by some Turing machine.

But here we are saying that it should be computable by polynomial time Turing machine and the rest is standard like suppose there is the function this from Σ^* to Σ^* and any w that is in A gets mapped to some w that is in B. And any w that is not in A gets mapped to some w that is not in B. So, A map to B and $\overline{\mathrm{A}}$ maps to $\overline{\mathrm{B}}$. So, the goal is given a situation, given a string that we do not know whether it is from A or whether it is an A or not, we want to transform it into the domain of B.

(Refer Slide Time: 04:09)



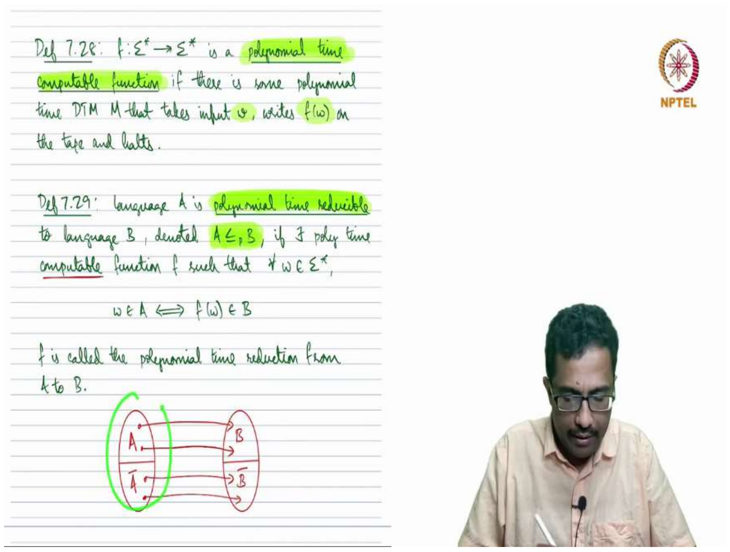
Def 7.28: $f:\Sigma^*\to\Sigma^*$ is a polynomial time computable function if there is some polynomial time DTM M that takes input w, writes f(w) on the tape and halts.

Def 7.29: language A is polynomial time reducible to language B, denoted $A \le_p B$, if $\exists$ poly time computable function f such that $\forall w \in \Sigma^*$,

$$w \in A \iff f(w) \in B$$

f is called the polynomial time reduction from A to B.

A
$\overline{A}$
B
$\overline{B}$



Def 7.28: $f:\Sigma^*\to\Sigma^*$ is a polynomial time computable function if there is some polynomial time DTM M that takes input w, writes f(w) on the tape and halts.

Def 7.29: language A is polynomial time reducible to language B, denoted $A \le_p B$, if $\exists$ poly time computable function f such that $\forall w \in \Sigma^*$,

$$w \in A \iff f(w) \in B$$

f is called the polynomial time reduction from A to B.

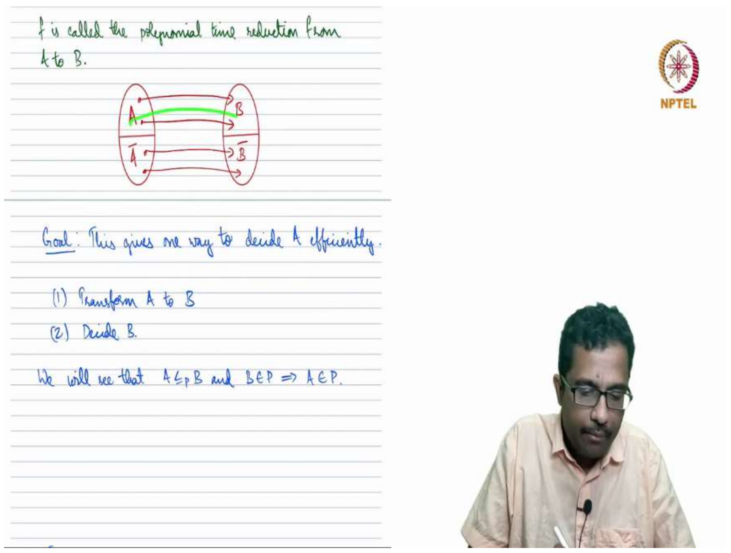
A
$\overline{A}$
B
$\overline{B}$

And suppose we know how to test the membership in B. So, we do not know how to test whether some string is in A or not, it could be in A or it could be in $\overline{\mathrm{A}}$, but we know how to test the membership in B then we can transform the instance, basically given w we can compute $f(w)$ and then check whether $f(w)$ is in B or not.

So, if $f(w)$ is in B, the only way that $f(w)$ could have been in B is if w was in A and if f(w) is in $\overline{\text{B}}$, the only way that we could come to $\overline{\text{B}}$ is if w is in $\overline{\text{A}}$. So, testing whether f(w) is in B or not is as good as testing whether w is in A or not. So that is the main goal of this reduction. But the main difference here from mapping reductions is that.

Now we are putting time constraints, or we are putting resource constraint on the how to compute the f, which is kind of not that surprising if you think about it, because we are in the chapter that pertains to time complexity. And we are dealing with complexity classes that do calculation as per time complexity, so we need some handle on how much time it takes, otherwise, this could become meaningless.

(Refer Slide Time: 05:46)



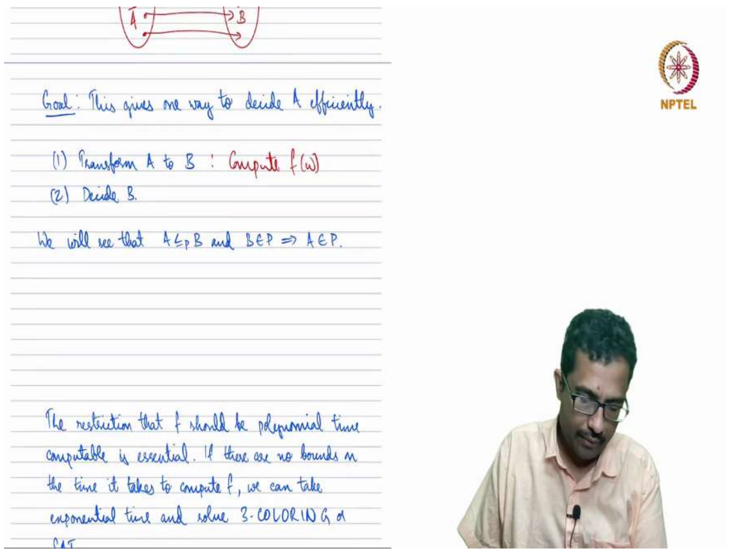
f is called the polynomial time reduction from A to B.

Goal: This gives me way to decide A efficiently.

(1) Transform A to B

(2) Decide B.

We will see that $A \leq_P B$ and $B \in P \Rightarrow A \in P$.



Goal: This gives me way to decide A efficiently.

(1) Transform A to B : Compute f(w)

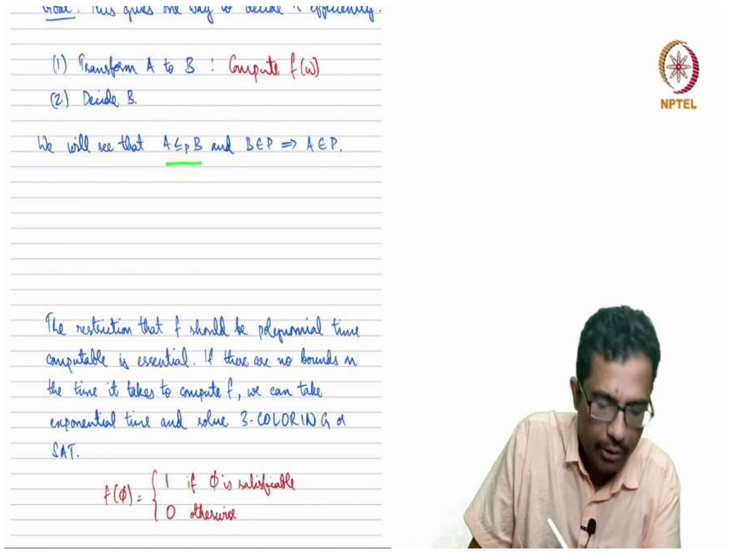
(2) Decide B.

We will see that $A \leq_P B$ and $B \in P \Rightarrow A \in P$.

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or

So, I will explain that in a bit. So, the goal as I mentioned is, suppose we are given some w. And we want to determine whether it is in A. So, the way to do it is we transform it to B we compute the function. And then check whether the function is in B or not. So, transform A to B means compute f(w) given w and then use the decider of B to decide B.

(Refer Slide Time: 06:16)



Goal: This gives me way to decide A efficiently.

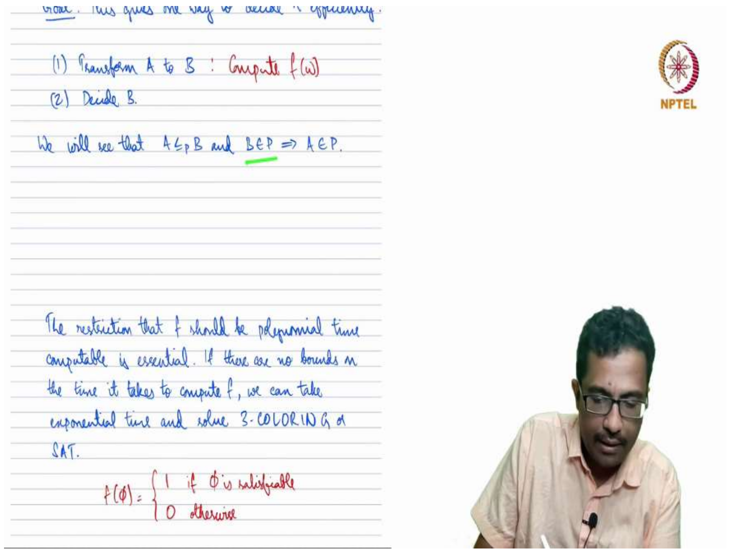
(1) Transform A to B : Compute $f(w)$

(2) Decide B.

We will see that $\underline{A \leq_p B}$ and $B \in P \Rightarrow A \in P$.

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or SAT.

$$f(\phi) = \begin{cases} 1 & \text{if } \phi \text{ is satisfiable} \\ 0 & \text{otherwise} \end{cases}$$



Goal: This gives me way to decide A efficiently.

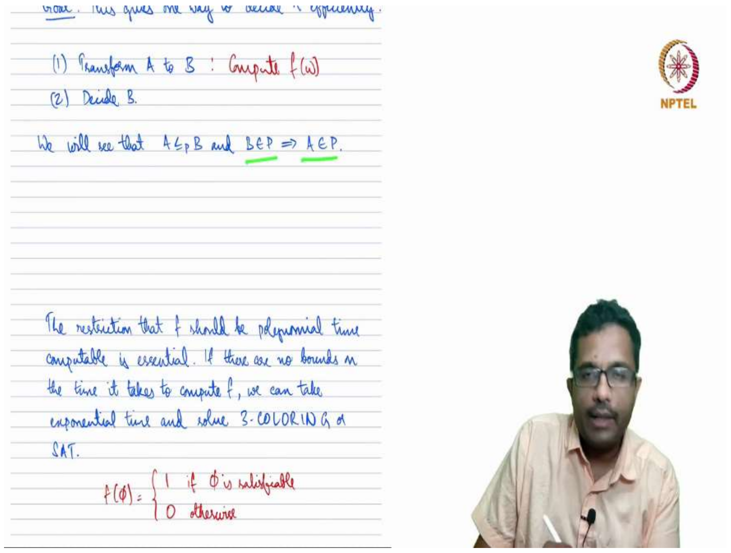
(1) Transform A to B : Compute $f(w)$

(2) Decide B.

We will see that $A \leq_p B$ and $\underline{B \in P} \Rightarrow A \in P$.

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or SAT.

$$f(\phi) = \begin{cases} 1 & \text{if } \phi \text{ is satisfiable} \\ 0 & \text{otherwise} \end{cases}$$



Goal: This gives one way to decide A efficiently.

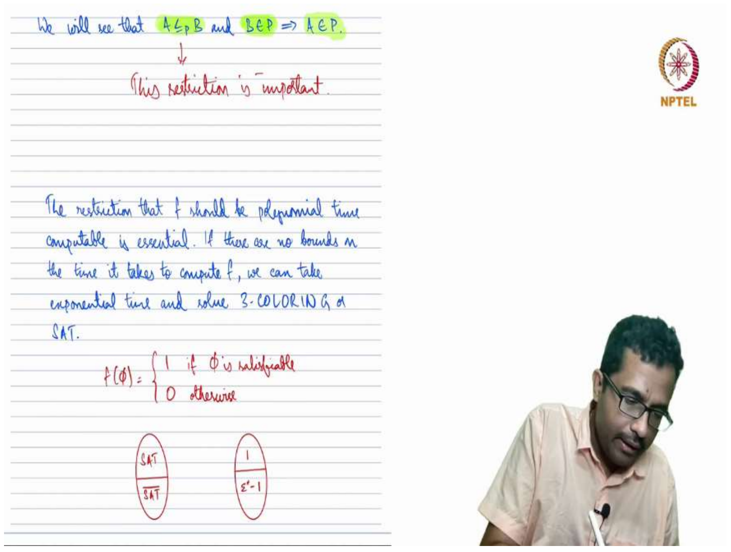
(1) Transform A to B : Compute $f(w)$

(2) Decide B.

We will see that $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or SAT.

$$f(\phi) = \begin{cases} 1 & \text{if } \phi \text{ is satisfiable} \\ 0 & \text{otherwise} \end{cases}$$



We will see that $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

This restriction is important.

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or SAT.

$$f(\phi) = \begin{cases} 1 & \text{if } \phi \text{ is satisfiable} \\ 0 & \text{otherwise} \end{cases}$$

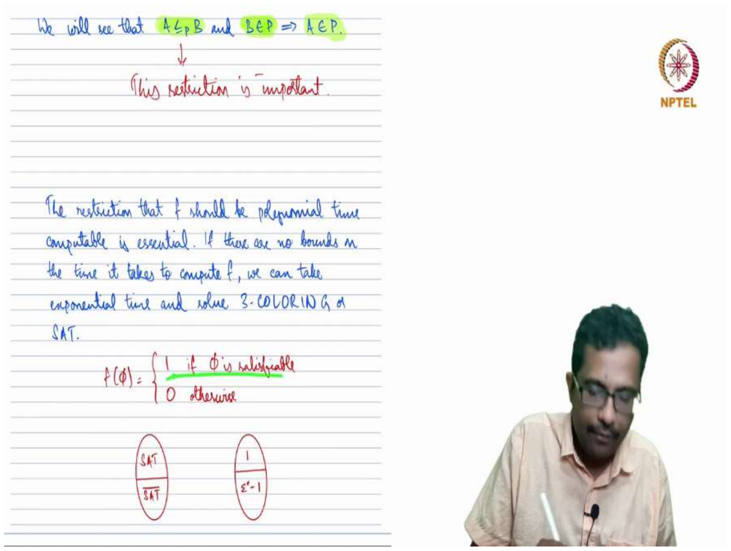
So, so later in a bit, we will see that we will see the following that suppose A is reducible to B in polynomial time, and suppose B is in P, this will imply that A is in P. So, this is going to be one of our main results that we will use suppose A is reducible to B and B is in P, then A is also in P. This is so, if you recall, we had this statement in chapter 5 that suppose A mapping reducible to B, and then B is decidable, this implies A is decidable.

This is something analogous to that. And the point that I want to say is that this restriction is important. So, suppose we did not have this restriction of polynomial time computable. Then let us say we could take exponential time to compute this function. Then the problem is that let us say we are given a three colouring instance or a satisfiable instance.

Then if I do not have any time bound, then I could try out all the possible 2^n assignments, let us say this is satisfiability. So, I will tell you how to compute a reduction to a language and the testing of the second language will be polynomial time, but the original language is something that we do not know whether it is in polynomial time or not.

So, suppose we have a SAT instance, what I can do is to compute exhaustively, all the possible assignments, and then basically decide SAT and then I output the function value.

(Refer Slide Time: 08:16)



We will see that $A \le_p B$ and $B \in P \Rightarrow A \in P$.

$\downarrow$

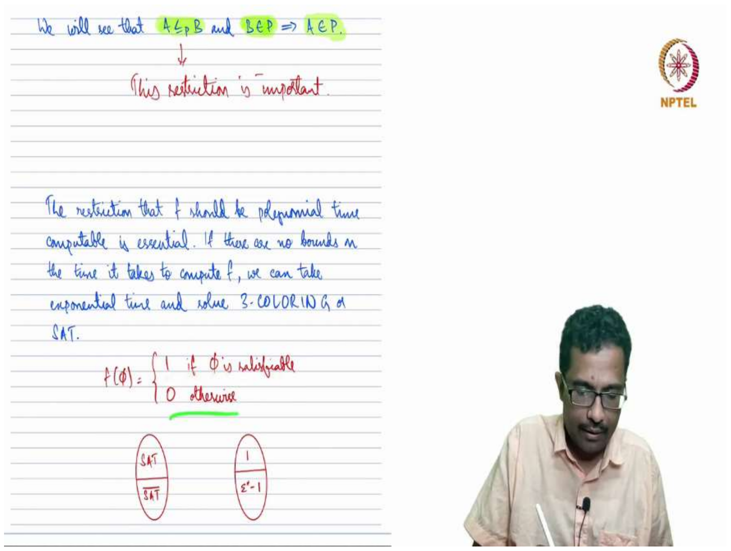
This restriction is important.

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or SAT.

$$f(\phi) = \begin{cases} 1 & \text{if } \phi \text{ is satisfiable} \\ 0 & \text{otherwise} \end{cases}$$

SAT | $\overline{\text{SAT}}$

1 | $\Sigma^* - 1$



We will see that $A \le_p B$ and $B \in P \Rightarrow A \in P$.

$\downarrow$

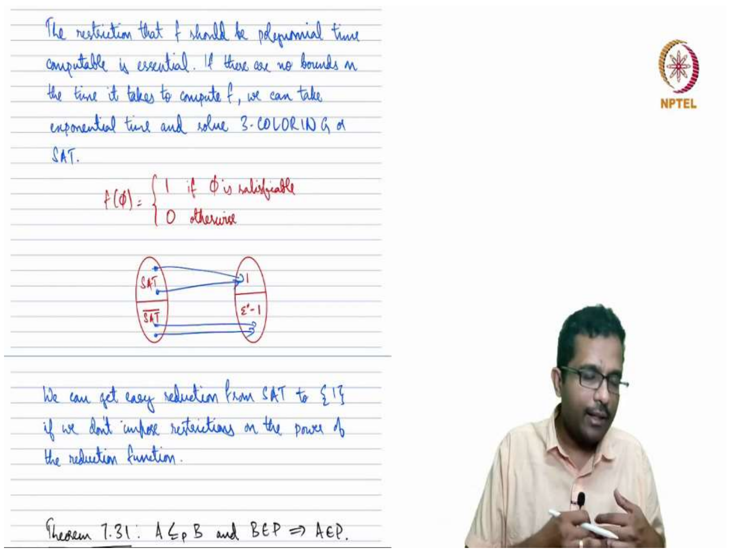
This restriction is important.

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or SAT.

$$f(\phi) = \begin{cases} 1 & \text{if } \phi \text{ is satisfiable} \\ 0 & \text{otherwise} \end{cases}$$

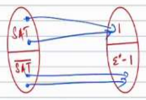
SAT | $\overline{\text{SAT}}$

1 | $\Sigma^* - 1$



The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or SAT.

$$f(\phi) = \begin{cases} 1 & \text{if } \phi \text{ is satisfiable} \\ 0 & \text{otherwise} \end{cases}$$



We can get easy reduction from SAT to {1} if we don't impose restrictions on the power of the reduction function.

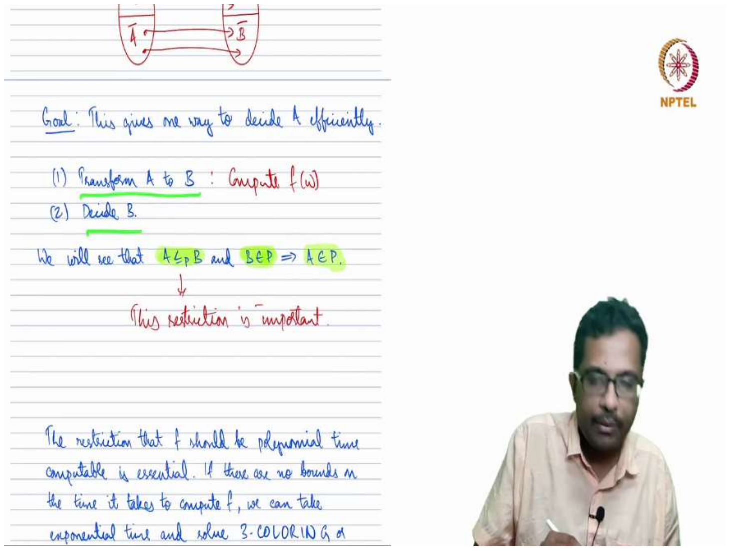
Theorem 7.31: $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

So the function value is going to be 1 if the formula is satisfiable, and 0 otherwise. So, I just take it 2^n time and try out all possible assignments and then basically have decided whether it is in satisfiable or not, but since my goal is to compute the reduction, I will output 1 if it is satisfiable, and 0 otherwise.

So now, this gives us a reduction, this gives us a reduction, although a very silly reduction. So, any, any, any formula that is satisfiable will map to 1 and any formula that is not satisfiable will map to some other string, will map to 0. So, basically, this is a reduction from satisfiability to the set one and everything that is not 1.

And then this is easy to decide, it is easy to decide whether suppose the transformed instance is quite easy to decide. So, now, you see that see our goal was to efficiently decide the language a. So now, so which means all the steps should be efficient, I should be able to efficiently compute the transformation and efficiently decide B.

(Refer Slide Time: 09:37)



Goal: This gives one way to decide A efficiently.

(1) Transform A to B : Compute $f(w)$

(2) Decide B.

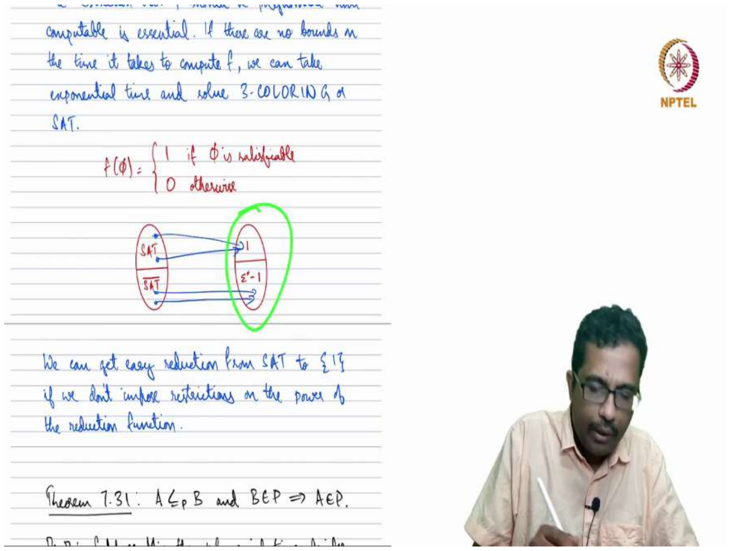
We will see that $A \leq_P B$ and $B \in P \Rightarrow A \in P$.

This restriction is important.

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or

So basically, both the steps here the transformation from A to B, this should be efficient, and the decision of B should be efficient. Now, if this transformation is not efficient, if I am allowed in finite time, then the whole point is not really meaningful.

(Refer Slide Time: 10:04)



computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or SAT.

$$f(\phi) = \begin{cases} 1 & \text{if } \phi \text{ is satisfiable} \\ 0 & \text{otherwise} \end{cases}$$

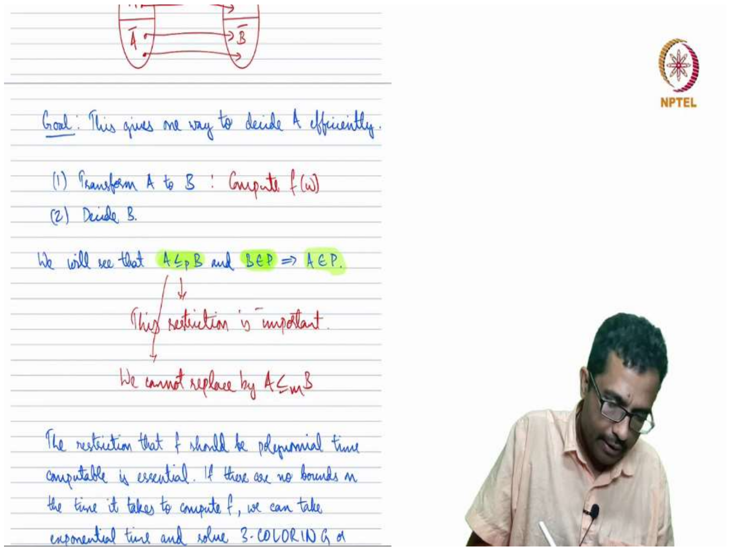
We can get easy reduction from SAT to $\{1\}$ if we don't impose restrictions on the power of the reduction function.

Theorem 7.31: $A \leq_P B$ and $B \in P \Rightarrow A \in P$.

So, like we have here, if I am not restricted in the time, I could easily compute the reduction from SAT to some trivial language like 1 and 0. And it is easy to decide this, easy to decide between 1 and 0. So, now, we have reduced SAT to a language that is easy to decide, but then the reduction takes exponential time.

So, we cannot basically hide the complexity of the problem in the reduction that is why, to make this whole thing meaningful, we have to impose a time restriction on the time that is taken for the reduction. So otherwise, it leads to such meaningless results. So, now, if you impose its time restrictions, then we cannot do this reduction from SAT to like the 0, 1 language. So, that is a point here.

(Refer Slide Time: 10:56)


$\bar{A}$ $\bar{B}$

Goal: This gives me way to decide A efficiently.

(1) Transform A to B : Compute $f(w)$
(2) Decide B.

We will see that $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

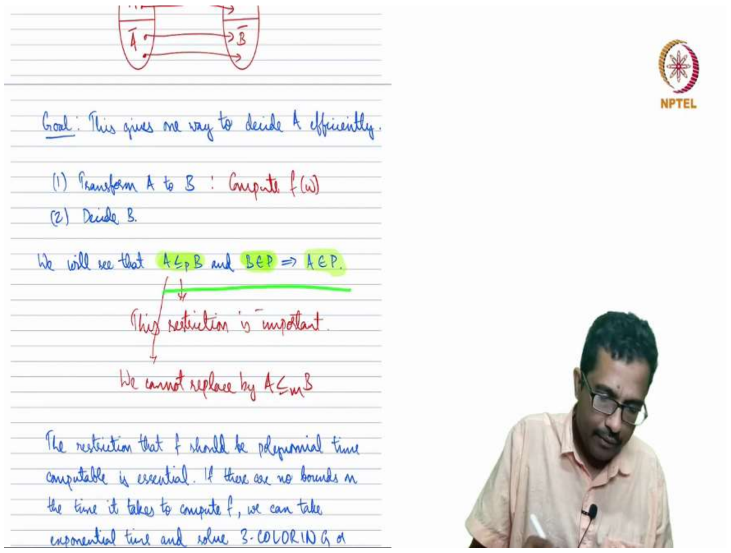
This restriction is important.

We cannot replace by $A \leq_m B$

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING G or

So, we for instance, we cannot replace this by a mapping reduction because then we cannot infer that A is in P, because the reduction itself may take exponential time. So, then how can we infer that the whole process is in polynomial time?

(Refer Slide Time: 11:28)



$\overline{A}$ $\overline{B}$

Goal: This gives one way to decide A efficiently.

(1) Transform A to B : Compute $f(w)$

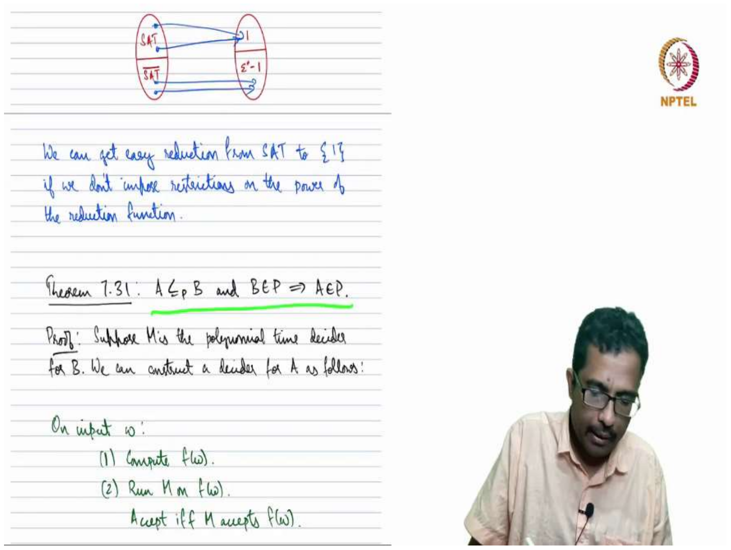
(2) Decide B.

We will see that $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

This restriction is important.

We cannot replace by $A \leq_m B$

The restriction that f should be polynomial time computable is essential. If there are no bounds on the time it takes to compute f, we can take exponential time and solve 3-COLORING or



SAT $\overline{SAT}$ 1 Σ^*-1

We can get easy reduction from SAT to {1} if we don't impose restrictions on the power of the reduction function.

Theorem 7.31: $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

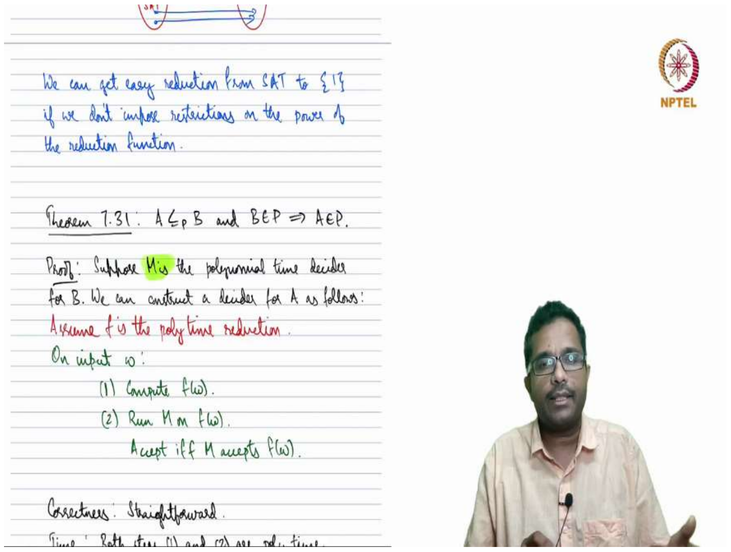
Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

On input w:

(1) Compute $f(w)$.

(2) Run M on $f(w)$.

Accept iff M accepts $f(w)$.



We can get easy reduction from SAT to {1} if we don't impose restrictions on the power of the reduction function.

Theorem 7.31: $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

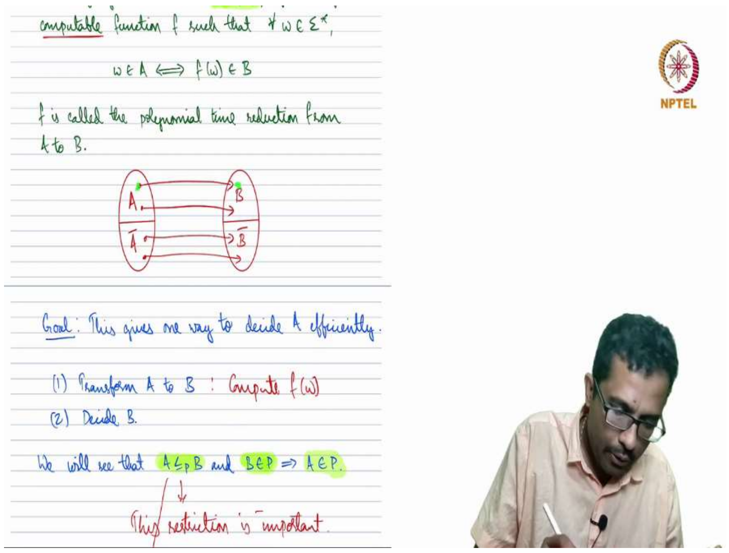
(1) Compute $f(w)$.

(2) Run M on $f(w)$.

Accept iff M accepts $f(w)$.

Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.


computable function f such that $\forall w \in \Sigma^*$,

$w \in A \iff f(w) \in B$

f is called the polynomial time reduction from A to B.

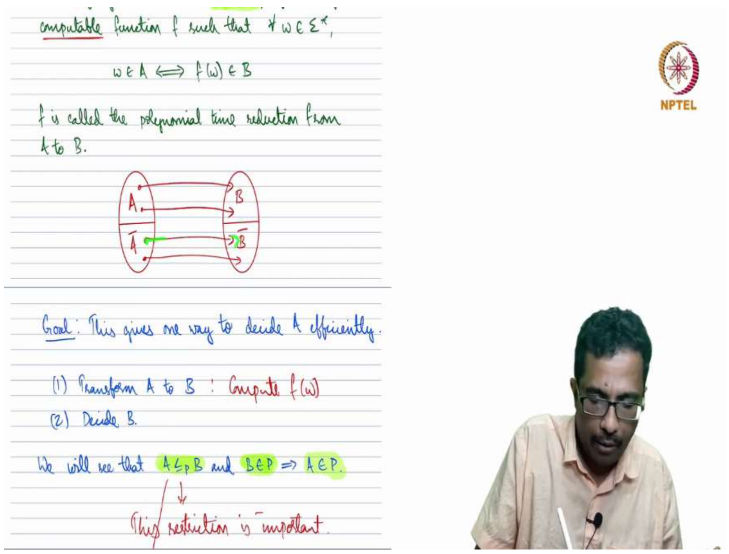
Goal: This gives me way to decide A efficiently.

(1) Transform A to B : Compute $f(w)$

(2) Decide B.

We will see that $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

This restriction is important.


computable function f such that $\forall w \in \Sigma^*$,

$w \in A \iff f(w) \in B$

f is called the polynomial time reduction from A to B.

Goal: This gives me way to decide A efficiently.

(1) Transform A to B : Compute $f(w)$

(2) Decide B.

We will see that $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

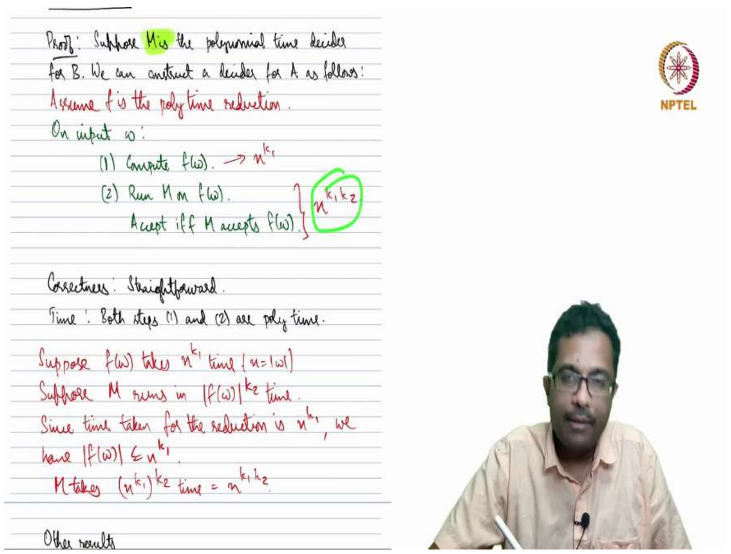
This restriction is important.

So now, let us come to the proof of the statement that that I mentioned over here that A reducible to B in polynomial time and B is in polynomial time implies A is in polynomial time. So, the idea is whatever we have already discussed. So, suppose M is the decider for B, M is the decider for B in polynomial time, we can construct a decider for A as follows, we first compute the function, the reduction from A to B.

So, f is the reduction, assume f is the reduction, maybe I will just write that assume f is the poly time reduction. So, we first compute the reduction f and then we check for the membership in B. So, we use the decider for B, we basically run M on the reduce instance, which is f(w).

And that is it and then if M accepts f(w) we accept w, if M rejects f(w) we reject. So, the correctness is straightforward, because only strings in A will come B and these strings will be accepted and those strings with $\overline{A}$ will come to $\overline{B}$ and those will be rejected. So, the correctness is straightforward. So, time complexity, let us just check the time complexity.

(Refer Slide Time: 12:56)



Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

(1) Compute f(w). $\rightarrow n^{k_1}$

(2) Run M on f(w).

Accept iff M accepts f(w). $\Big\}\ \boxed{n^{k_1 k_2}}$

Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

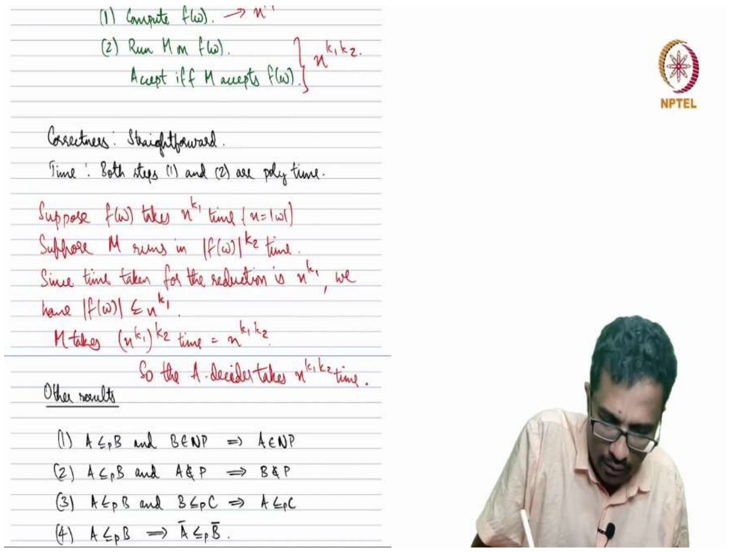
Suppose f(w) takes n^{k_1} time ($n = |w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$

Other results



(1) Compute f(w). $\rightarrow n^{k_1}$

(2) Run M on f(w).

Accept iff M accepts f(w). $\Big\}\ n^{k_1 k_2}$.

Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose f(w) takes n^{k_1} time ($n = |w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$

So the A-decider takes $n^{k_1 k_2}$ time.

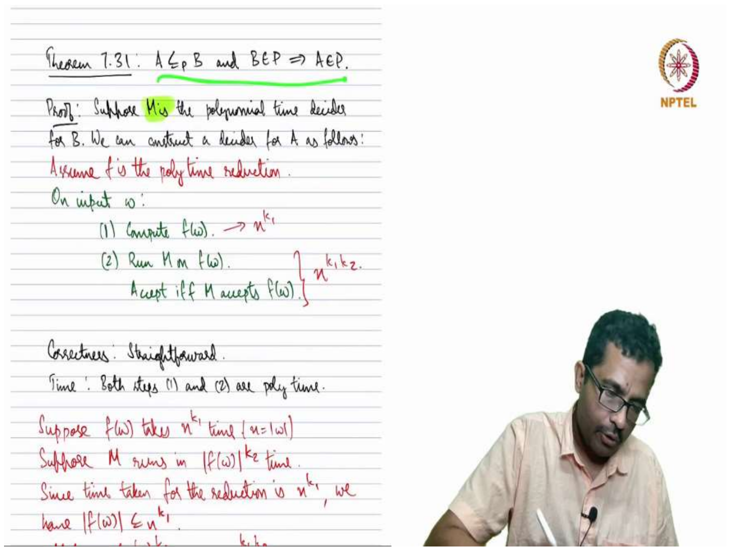
Other results

(1) $A \leq_p B$ and $B \in NP \Rightarrow A \in NP$

(2) $A \leq_p B$ and $A \notin P \Rightarrow B \notin P$

(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$

(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.



Theorem 7.31: $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

(1) Compute f(w). $\rightarrow n^{k_1}$

(2) Run M on f(w).

Accept iff M accepts f(w). $\Big\}\ n^{k_1 k_2}$.

Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose f(w) takes n^{k_1} time ($n = |w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

Suppose to compute f(w) takes let us say n^{k1} time which means this step takes the computing f(w) takes n^{k1} time, where let us say n is the length of the input. Now, let us say suppose M takes or M runs in, whatever is the input length to the power k2.

So, the input to M is f(w). Suppose the exponent of M is k2. We know M is a polynomial time Turing machine. So, it will run in the length power k2 for some constant k2. But what is the length of f(w)? So, we know only the length of w. What is the length of f(w)? So thing is that, the only thing that we know about f(w) is that it is produced by a Turing machine that runs in time n^{k1}.

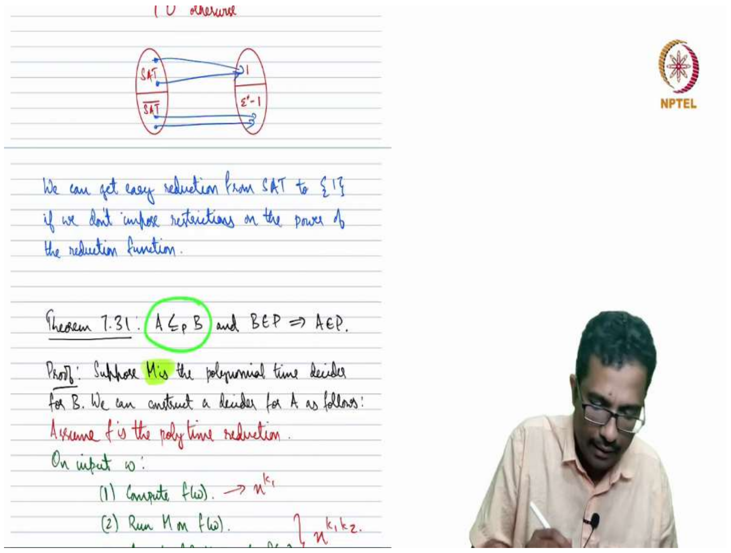
So, which means in that time, it even has to write the output. So, the length of f(w) cannot be more than in n^{k1}. If it is more, then writing the output will take more time. So, we can say that since time taken for the reduction is n^{k1}. We have the length of f(w) also should be at most n^{k1} which means M takes $(n^{k1})^{k2}$ time which is n^{k1*k2}.

So, this this step takes n^{k1*k2}. So, the total time taken here is $n^{k1} + n^{k1*k2}$. So, the second part is that this is the dominating thing n^{k1*k2} which is okay but the exponent is still a constant. So, n^{k1*k2} is still polynomial in the input length where the input is the string w. So, the decider takes n^{k1*k2} time.

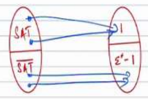
So, this is polynomial time, polynomial in the length of the input. So, that is the proof. So, the proof is very simple we convert the A instance to B and then we run the B decider on the reduced instance and the correctness is quite clear and the running time for the whole process is the running time for the conversion which is n^{k1} and the running time for running the decider on the reduced instance which is the length of the reduced instance power k2 which turns out to be n^{k1*k2} .

So, the whole running time is $n^{k1} + n^{k1*k2}$ which is dominated by n^{k1*k2} which is a polynomial time in the length of the input. So, hence A is in P. So, we have given a decider for A which runs in polynomial time and what I said before. So, this proof is complete.

(Refer Slide Time: 17:47)



0 otherwise



We can get easy reduction from SAT to $\{1\}$ if we don't impose restrictions on the power of the reduction function.

Theorem 7.31: $\boxed{A \leq_p B}$ and $B \in P \Rightarrow A \in P$.

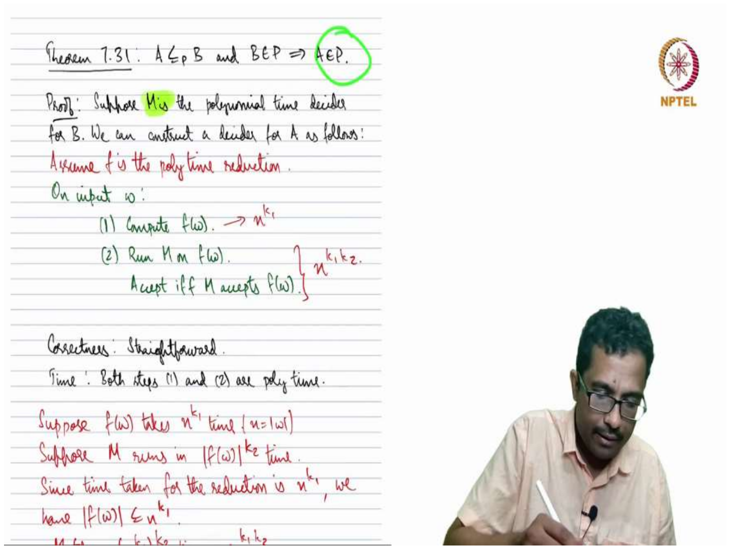
Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

(1) Compute f(w). $\rightarrow n^{k_1}$

(2) Run M on f(w). $\Big\} n^{k_1 k_2}$.



Theorem 7.31: $A \leq_p B$ and $B \in P \Rightarrow \boxed{A \in P}$.

Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

(1) Compute f(w). $\rightarrow n^{k_1}$

(2) Run M on f(w).

Accept iff M accepts f(w). $\Big\} n^{k_1 k_2}$.

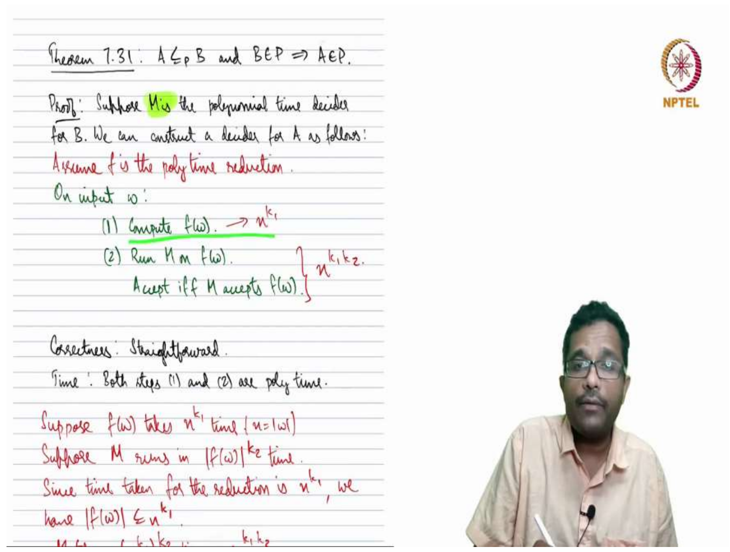
Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose f(w) takes n^{k_1} time ($n = |w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.



Theorem 7.31: $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

(1) $\underline{\text{Compute f(w).} \rightarrow n^{k_1}}$

(2) Run M on f(w).

Accept iff M accepts f(w). $\Big\} n^{k_1 k_2}$.

Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

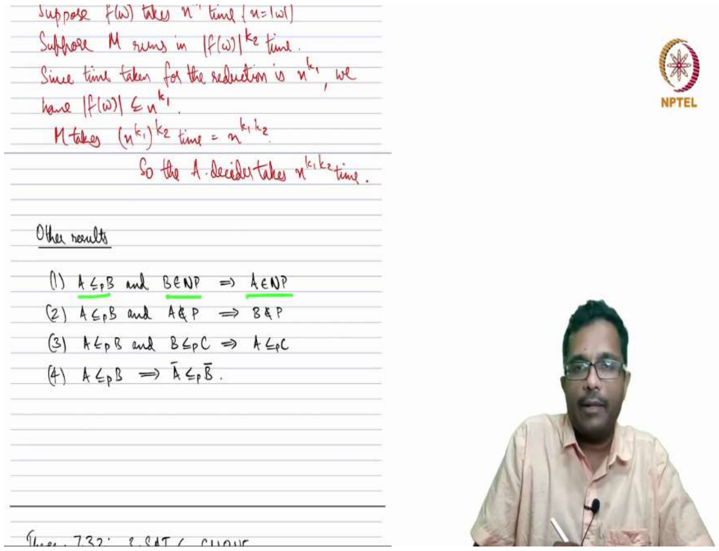
Suppose f(w) takes n^{k_1} time ($n = |w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

What I said before is that if we did not have this restriction that the reduction from A to B is in polynomial time, we cannot infer this because the first step itself would have taken more time than polynomial.

(Refer Slide Time: 18:19)



Suppose $f(w)$ takes n^{k_1} time ($n=|w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$.

So the A-decider takes $n^{k_1 k_2}$ time.

Other results

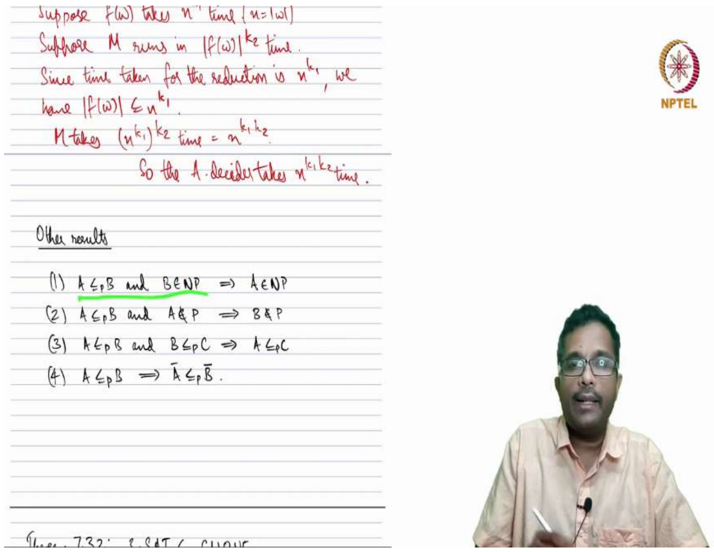
(1) $\underline{A \leq_p B}$ and $\underline{B \in NP}$ $\Rightarrow$ $\underline{A \in NP}$

(2) $A \leq_p B$ and $A \notin P$ $\Rightarrow$ $B \notin P$

(3) $A \leq_p B$ and $B \leq_p C$ $\Rightarrow$ $A \leq_p C$

(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.

Theo. 7.32: 3SAT ≤ CLIQUE



Suppose $f(w)$ takes n^{k_1} time ($n=|w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$.

So the A-decider takes $n^{k_1 k_2}$ time.

Other results

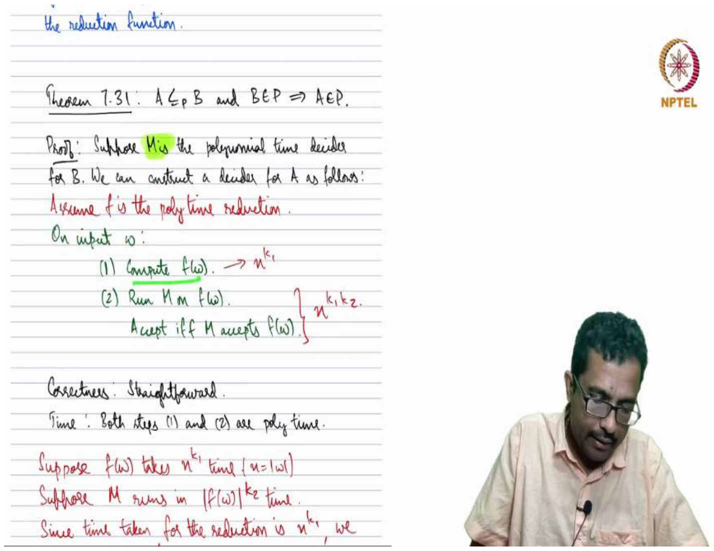
(1) $\underline{A \leq_p B \text{ and } B \in NP}$ $\Rightarrow$ $A \in NP$

(2) $A \leq_p B$ and $A \notin P$ $\Rightarrow$ $B \notin P$

(3) $A \leq_p B$ and $B \leq_p C$ $\Rightarrow$ $A \leq_p C$

(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.

Theo. 7.32: 3SAT ≤ CLIQUE



the reduction function.

Theorem 7.31: $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

(1) Compute f(w). $\rightarrow n^{k_1}$

(2) Run M on f(w).
Accept iff M accepts f(w). $\Big\}\ n^{k_1 k_2}$.

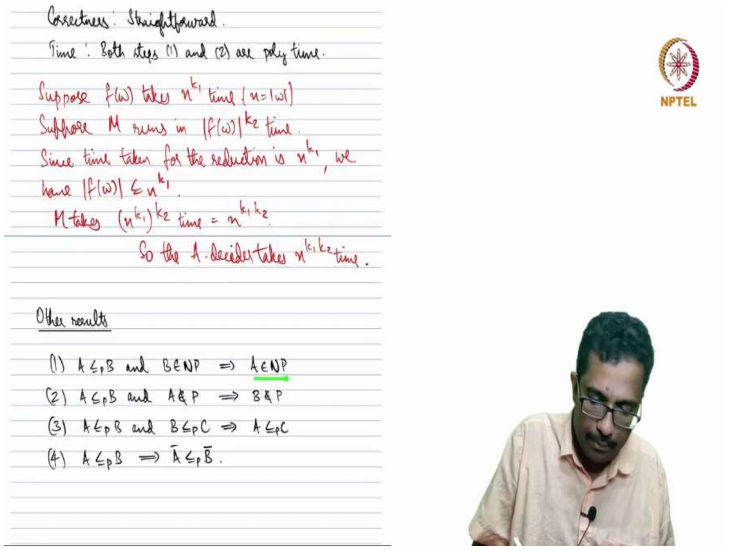
Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n=|w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we


Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n=|w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$

So the A-decider takes $n^{k_1 k_2}$ time.

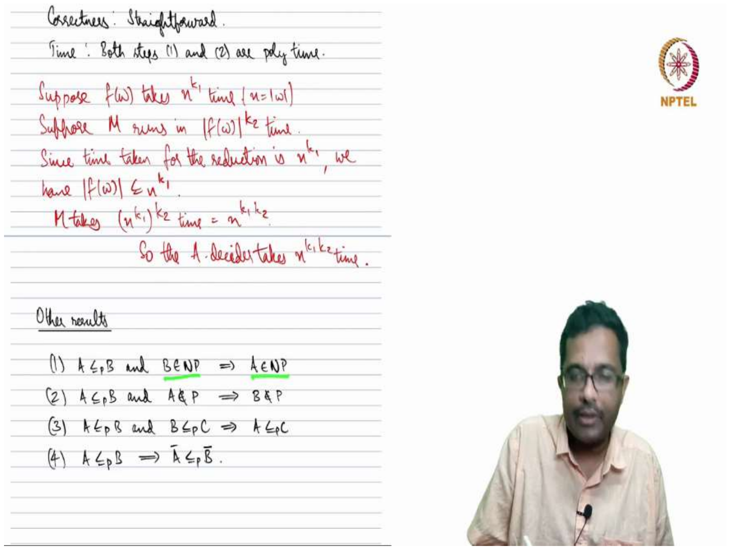
Other results

(1) $A \leq_p B$ and $B \in NP \Rightarrow \underline{A \in NP}$

(2) $A \leq_p B$ and $A \notin P \Rightarrow B \notin P$

(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$

(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.


Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n=|w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$

So the A-decider takes $n^{k_1 k_2}$ time.

Other results

(1) $A \leq_p B$ and $\underline{B \in NP} \Rightarrow \underline{A \in NP}$

(2) $A \leq_p B$ and $A \notin P \Rightarrow B \notin P$

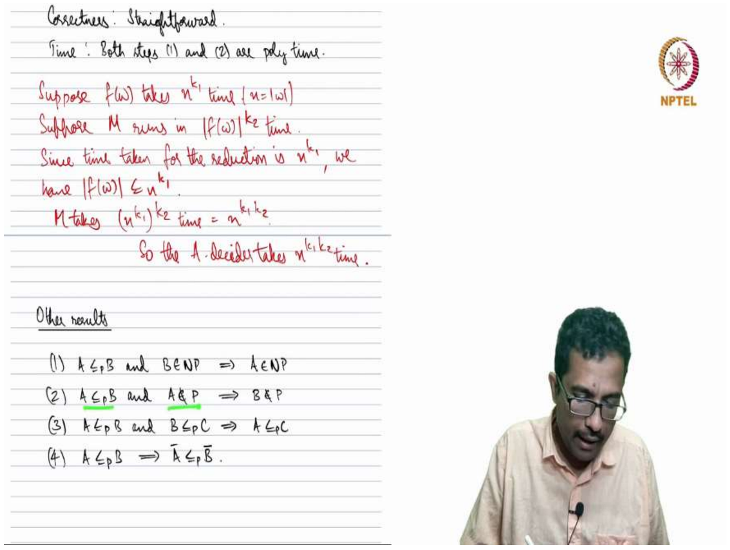
(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$

(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.

Now, briefly some other results that I just mentioned some other results that I will mention. One is that the same thing, if A reduces to B in polynomial time and B is in NP, then A is in NP this is true because. So, we need so, if A reduces to B and B is in NP, then A is an NP. So, basically, we these two should together imply an NP decider for a which is straightforward by the same process.

So, we compute the reduction and then run the non-deterministic decider for B on the reduced instance. So, that together they get a non-deterministic decider for A. So, the claim is that A is in NP. So, if B is in NP, then that gives an non deterministic decider for A.

(Refer Slide Time: 19:00)


Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n = |w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$.

So the A-decider takes $n^{k_1 k_2}$ time.

<u>Other results</u>

(1) $A \leq_p B$ and $B \in NP \Rightarrow A \in NP$

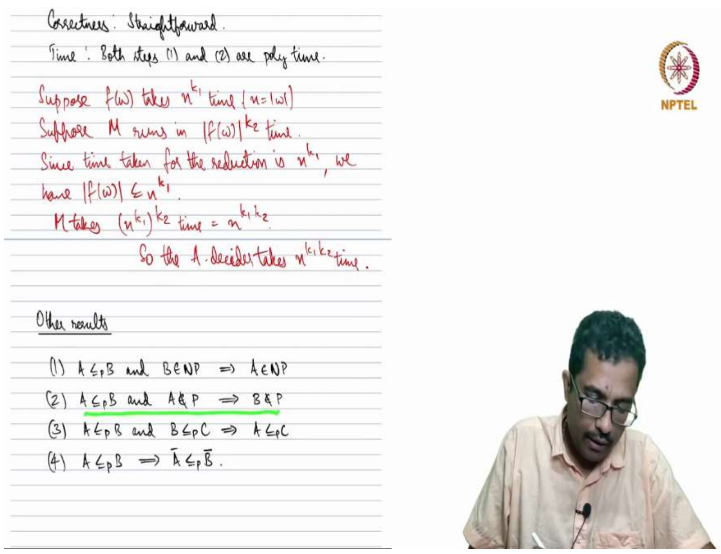
(2) $\underline{A \leq_p B}$ and $\underline{A \notin P} \Rightarrow B \notin P$

(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$

(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.

The next thing is that if A reduces to B and A is not in P then B is not in P. So, the analogous result here is what we saw in chapter 5 if A was mapping reducible to B and A was undecidable, then B is undecidable. Why is this true?

(Refer Slide Time: 19:30)



Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n=|w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \le n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$

So the A-decider takes $n^{k_1 k_2}$ time.

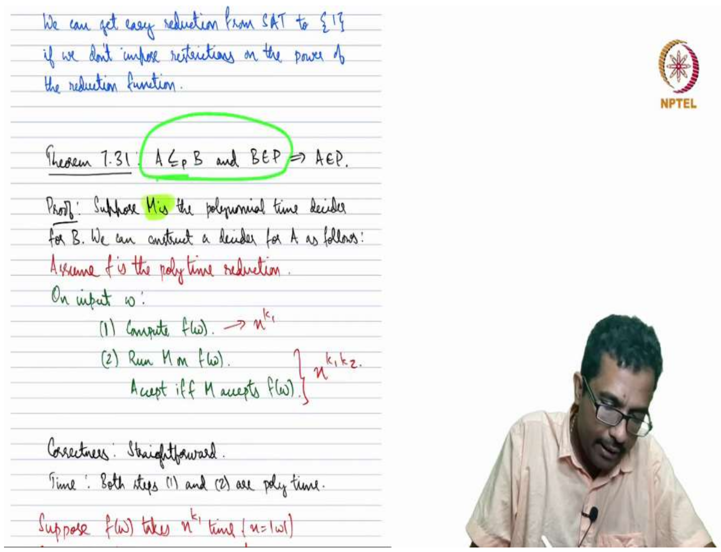
Other results

(1) $A \le_p B$ and $B \in NP \Rightarrow A \in NP$

(2) $A \le_p B$ and $A \notin P \Rightarrow B \notin P$

(3) $A \le_p B$ and $B \le_p C \Rightarrow A \le_p C$

(4) $A \le_p B \Rightarrow \bar{A} \le_p \bar{B}$.



We can get easy reduction from SAT to $\{1\}$ if we dont impose restrictions on the power of the reduction function.

Theorem 7.31: $A \le_p B$ and $B \in P \Rightarrow A \in P$.

Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

(1) Compute $f(w)$. $\rightarrow n^{k_1}$

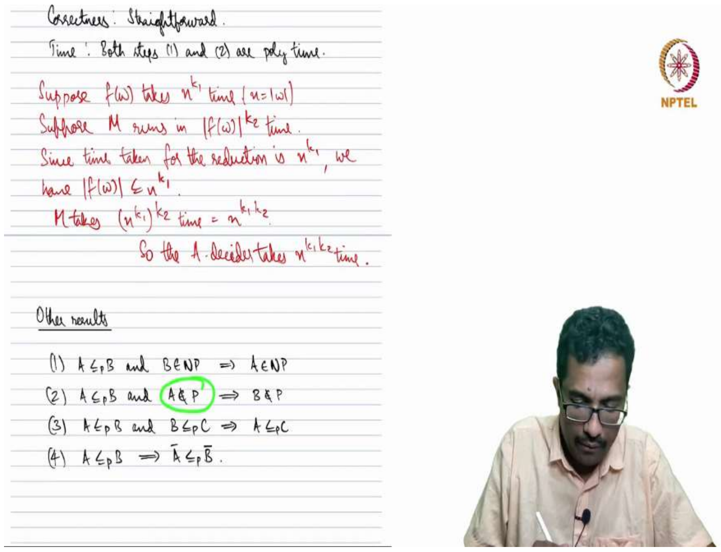
(2) Run M on $f(w)$.

Accept iff M accepts $f(w)$. $\Big\}\, n^{k_1 k_2}$

Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n=|w|$)



Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n=|w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \le n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$

So the A-decider takes $n^{k_1 k_2}$ time.

Other results

(1) $A \le_p B$ and $B \in NP \Rightarrow A \in NP$

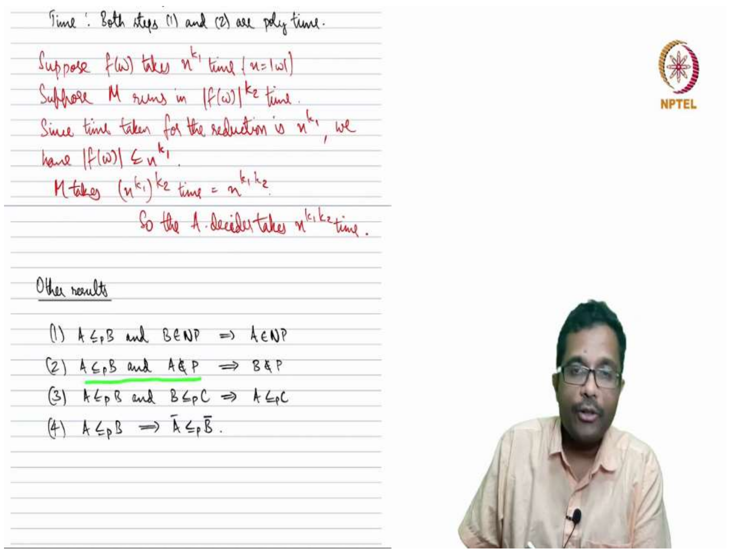
(2) $A \le_p B$ and $A \notin P \Rightarrow B \notin P$

(3) $A \le_p B$ and $B \le_p C \Rightarrow A \le_p C$

(4) $A \le_p B \Rightarrow \bar{A} \le_p \bar{B}$.

So, suppose this is not true if A reducible to B and A are not in P suppose B was in P. Suppose B was in P then by this theorem, it would imply that A is in P which is a contradiction. So, if A is not in P, then B should certainly be not in P. Because if B was in P, then that gives a way for us to decide A in polynomial time.

(Refer Slide Time: 20:01)



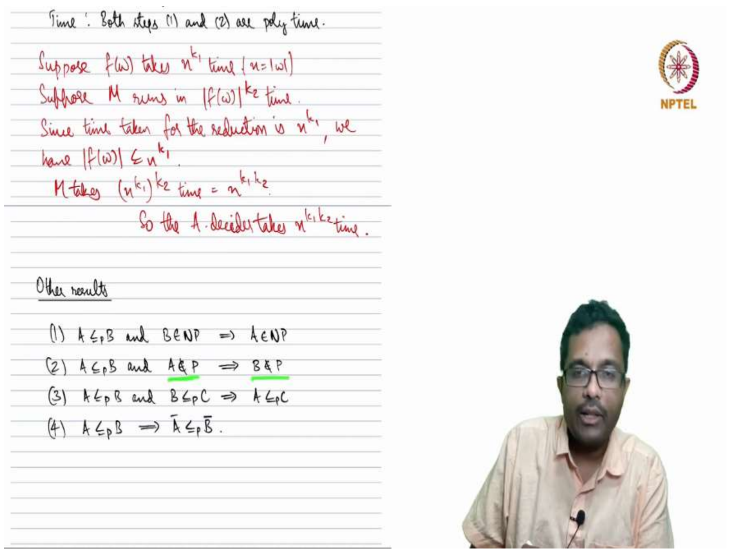
Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n = |w|$)
Suppose M runs in $|f(w)|^{k_2}$ time.
Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.
M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$

So the A decider takes $n^{k_1 k_2}$ time.

Other results

(1) $A \leq_p B$ and $B \in NP \Rightarrow A \in NP$
(2) $\underline{A \leq_p B \text{ and } A \notin P}$ $\Rightarrow B \notin P$
(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$
(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.



Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n = |w|$)
Suppose M runs in $|f(w)|^{k_2}$ time.
Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.
M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$

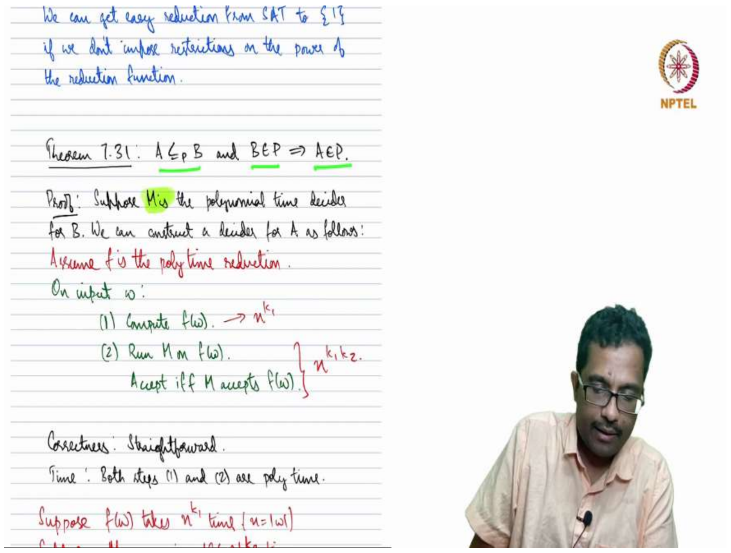
So the A decider takes $n^{k_1 k_2}$ time.

Other results

(1) $A \leq_p B$ and $B \in NP \Rightarrow A \in NP$
(2) $A \leq_p B$ and $\underline{A \notin P} \Rightarrow \underline{B \notin P}$
(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$
(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.

So, if A is reducible to B in polynomial time and A not in P that implies that B not in P.

(Refer Slide Time: 20:13)

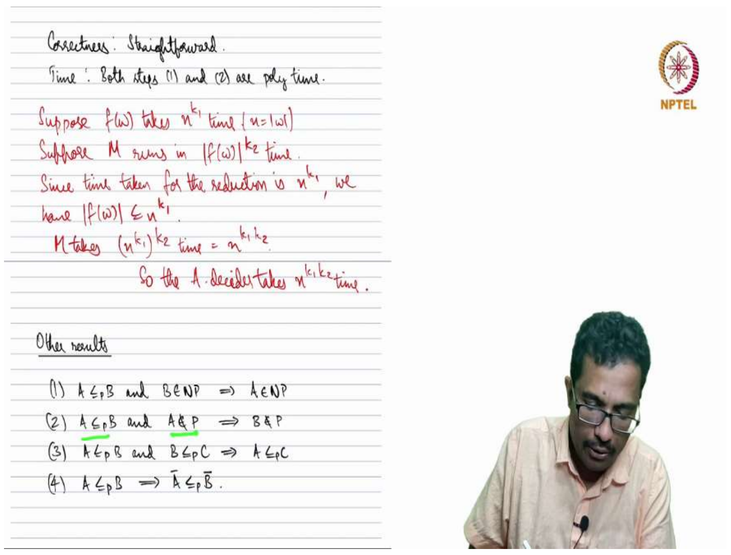

We can get easy reduction from SAT to $\{1\}$ if we don't impose restrictions on the power of the reduction function.

Theorem 7.31: $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:
Assume f is the poly time reduction.
On input w:
(1) Compute $f(w)$. $\rightarrow n^{k_1}$
(2) Run M on $f(w)$.
Accept iff M accepts $f(w)$. $\}\ n^{k_1 k_2}$.

Correctness: Straightforward.
Time: Both steps (1) and (2) are poly time.

Suppose $f(w)$ takes n^{k_1} time ($n = |w|$)


Correctness: Straightforward.
Time: Both steps (1) and (2) are poly time.

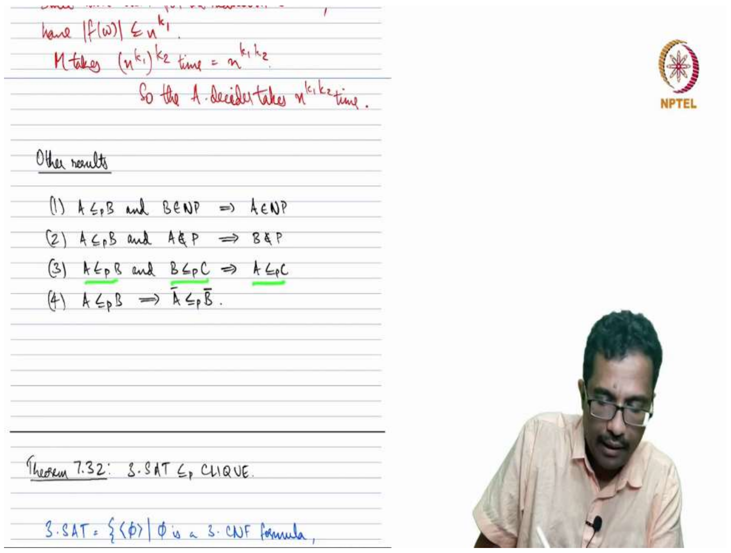
Suppose $f(w)$ takes n^{k_1} time ($n = |w|$)
Suppose M runs in $|f(w)|^{k_2}$ time.
Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.
M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$.
So the A decider takes $n^{k_1 k_2}$ time.

Other results

(1) $A \leq_p B$ and $B \in NP \Rightarrow A \in NP$
(2) $A \leq_p B$ and $A \notin P \Rightarrow B \notin P$
(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$
(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.

Again, just two analogous results were if A was mapping reducible to B and B was decidable, A was decidable. And the analogous result for this is that if A is mapping reducible to B and A is undecidable, then B is undecidable.

(Refer Slide Time: 20:31)



have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$.

So the A-decider takes $n^{k_1 k_2}$ time.

<u>Other results</u>

(1) $A \leq_p B$ and $B \in NP \Rightarrow A \in NP$

(2) $A \leq_p B$ and $A \notin P \Rightarrow B \notin P$

(3) $\underline{A \leq_p B}$ and $\underline{B \leq_p C} \Rightarrow \underline{A \leq_p C}$

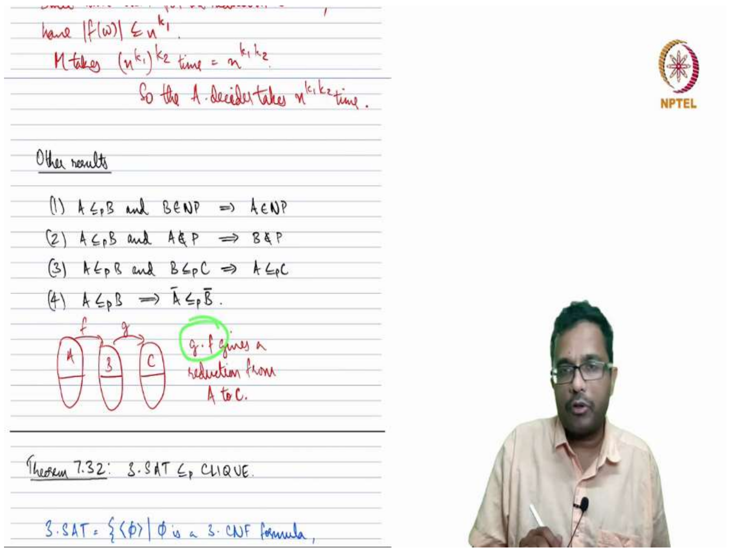
(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.

<u>Theorem 7.32</u>: 3-SAT $\leq_p$ CLIQUE.

3-SAT $= \{\langle \phi \rangle \mid \phi$ is a 3-CNF formula,

The next thing is transitivity. So, suppose A reduces to B and B reduces to C then A reduces to C. Why is this true?

(Refer Slide Time: 20:39)



have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$.

So the A-decider takes $n^{k_1 k_2}$ time.

Other results

(1) $A \leq_p B$ and $B \in NP \Rightarrow A \in NP$

(2) $A \leq_p B$ and $A \notin P \Rightarrow B \notin P$

(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$

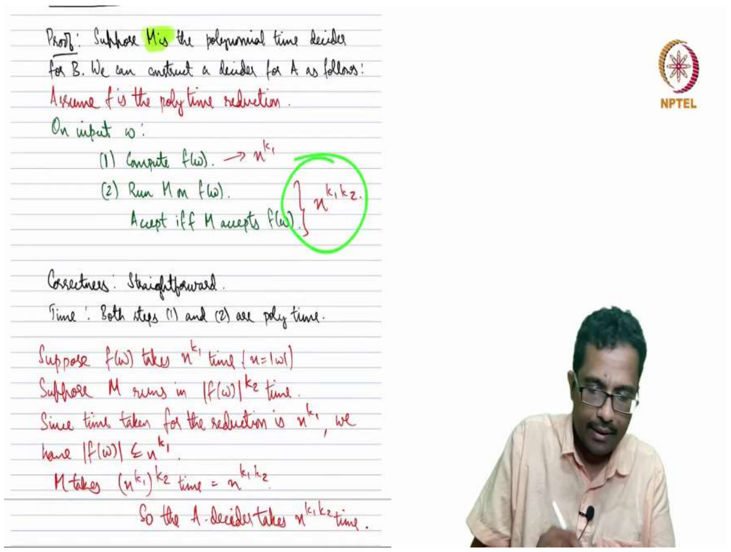
(4) $A \leq_p B \Rightarrow \bar{A} \leq_p \bar{B}$.

$$\begin{tikzcd} A \arrow[r, bend left, "f"] & B \arrow[r, bend left, "g"] & C \end{tikzcd}$$

$g \cdot f$ gives a reduction from A to C.

Theorem 7.32: 3-SAT $\leq_p$ CLIQUE.

3-SAT = $\{\langle \phi \rangle \mid \phi$ is a 3-CNF formula,



Proof: Suppose M is the polynomial time decider for B. We can construct a decider for A as follows:

Assume f is the poly time reduction.

On input w:

(1) Compute $f(w)$. $\rightarrow n^{k_1}$

(2) Run M on $f(w)$.

Accept iff M accepts $f(w)$. $\} n^{k_1 k_2}$.

Correctness: Straightforward.

Time: Both steps (1) and (2) are poly time.

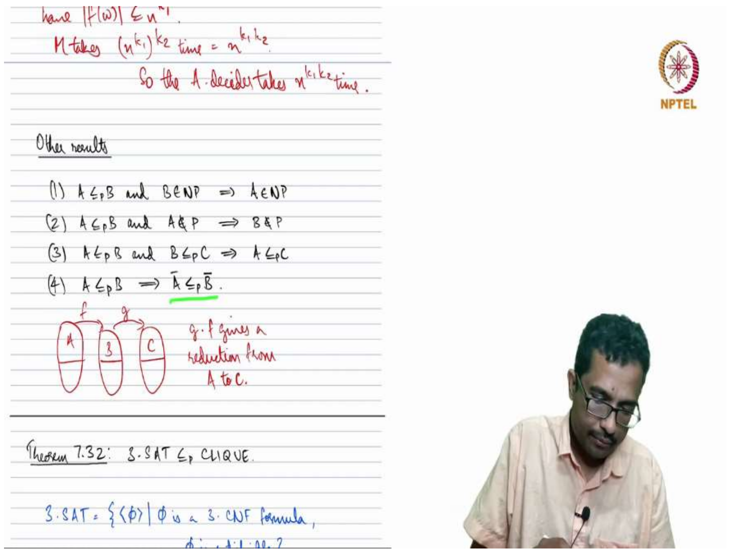
Suppose $f(w)$ takes n^{k_1} time ($n = |w|$)

Suppose M runs in $|f(w)|^{k_2}$ time.

Since time taken for the reduction is n^{k_1}, we have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$.

So the A-decider takes $n^{k_1 k_2}$ time.



have $|f(w)| \leq n^{k_1}$.

M takes $(n^{k_1})^{k_2}$ time $= n^{k_1 k_2}$.

So the A-decider takes $n^{k_1 k_2}$ time.

Other results

(1) $A \leq_p B$ and $B \in NP \Rightarrow A \in NP$

(2) $A \leq_p B$ and $A \notin P \Rightarrow B \notin P$

(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$

(4) $A \leq_p B \Rightarrow \underline{\bar{A} \leq_p \bar{B}}$.

$$\begin{tikzcd} A \arrow[r, bend left, "f"] & B \arrow[r, bend left, "g"] & C \end{tikzcd}$$

$g \cdot f$ gives a reduction from A to C.

Theorem 7.32: 3-SAT $\leq_p$ CLIQUE.

3-SAT = $\{\langle \phi \rangle \mid \phi$ is a 3-CNF formula,

Because, we could just take the composition of the two reductions. So, suppose this is A, B and C and the function f takes you from A to B and the function g takes you from B to C and we can compose these two function, we can take $g \circ f$ to compute a reduction from A to C. So, $g \circ f$ gives a reduction from A to C. Because, so, given w we compute f and then $g(f(w))$.

And the correctness is easy because anything that is in A goes to B in that goes to C and anything that is not in A goes to $\overline{\mathrm{B}}$ then goes to $\overline{\mathrm{C}}$. And the time complexity is also not that difficult to see basically we want to say that the time complexity of computing the composition is also polynomial in the length of the input. And that the reasoning for that is similar to the way we argued the time complexity of this theory, this n^{k1*k2}.

So, this you try out as an exercise, the correctness is evident, and the time complexity is similar to the proof of this theorem. And finally, one more thing that I want to say is if A reduces to B in polynomial time, $\overline{\mathrm{A}}$ reduces to $\overline{\mathrm{B}}$. Basically, the same reduction function does what we want, because anything from $\overline{\mathrm{A}}$ maps to $\overline{\mathrm{B}}$, anything from A map to B, same function serves as this reduction as well.

So, these are kind of standard facts, but, try it out and try to prove these things and try to write down this and prove it formally. I have given you the proof, at least orally I told you the proof, but to get some practice you may try to write down the proofs yourself. So, that completes, the definition of polynomial time reduction and some properties.