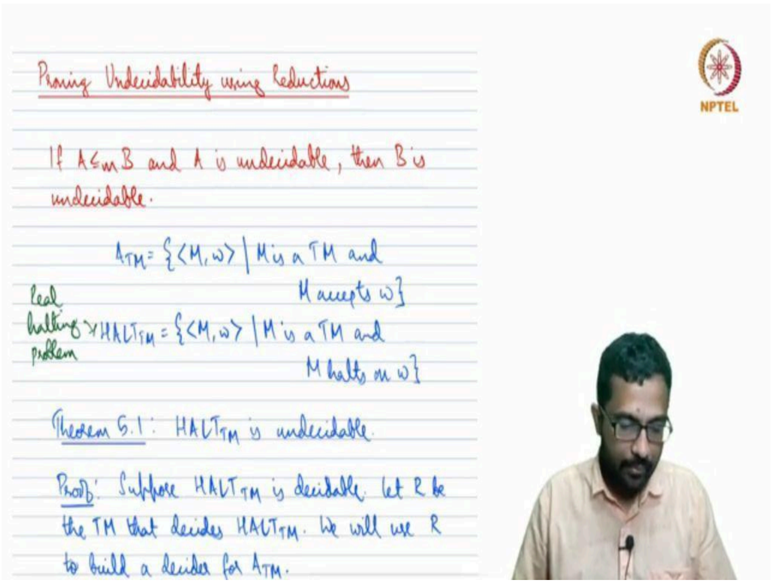
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Examples of Proving Undecidability Using Reductions

(Refer Slide Time: 00:16)



Proving Undecidability using Reductions

If $A \leq_m B$ and A is undecidable, then B is undecidable.

$A_{TM} = \{\langle M, w\rangle \mid M$ is a TM and M accepts $w\}$

Real halting problem $\rightarrow HALT_{TM} = \{\langle M, w\rangle \mid M$ is a TM and M halts on $w\}$

Theorem 5.1: $HALT_{TM}$ is undecidable.

Proof: Suppose $HALT_{TM}$ is decidable. Let R be the TM that decides $HALT_{TM}$. We will use R to build a decider for A_{TM}.

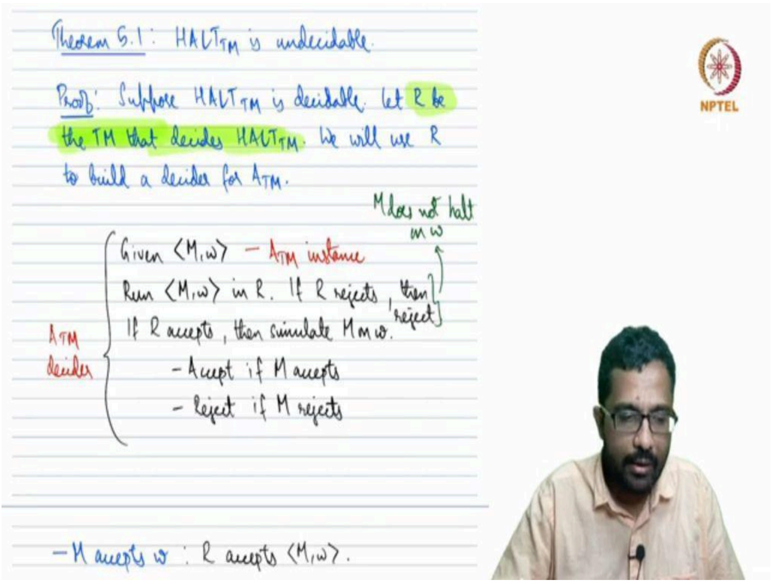
Hello and welcome to lecture 40 of the cost Theory of Computation. In the previous lecture, we saw what reductions are, we define mapping reducibility. And then we sought to, we made two statements. One is that if A is reducible A reduces to B and if B is decidable that A is an, A is decidable as well. So, you could use the reduction from A to B and then the decider for B, and combine them to get a decider for A. The other result that followed was that if A reduces to B and A is undecidable, then B is also undecidable. So, that is what I have written over here. If A reduces to B and A is undecidable, then B is also undecidable. And this is going to be very useful for us.

So, far the main undecidable language that we have seen that we have proved to be undecidable was A_TM. We prove from scratch using all the theory of countable and uncountable sets and all that. So, A_TM was the set of all Turing machines M and strings w, where M accepts w. In other words, we are asking given a Turing Machine M and a string w, we want to decide if M accepts w. So, now we are going to use the undecidability of A_TM and the fact that we can use the reductions, and using this theorem. We will combine all this to show the undecidability of other languages as well.

So, the first language that we will show undecidability is HALT_TM. So, what is HALT_TM? Given a Turing Machine M and a string w, we are asking whether this Turing machine M halts on w, meaning let M run on W. Does M halt on w or does M loop on W? This is the question. And sometimes this is referred to as a real halting problem, real halting problem in the sense that A_TM also sometimes referred to as the halting problem. But, here is a problem that we are actually checking whether it is halting or not. So, there is a reason why A_TM is also referred to as a halting problem. Because of the difficulty, so, you can always run M on the string w.

And this is we can just report the output, but the challenge comes when M does not halt on w at all. So, there is no way to tell whether it is going to halt or not. So, the challenge is in finding whether M halts on w even in A_TM. So, that is why we sometimes refer to A_TM also as the halting problem, because the real difficulty there also lies in the, lies in the issue of determining whether M halts on w.

(Refer Slide Time: 03:15)



Theorem 5.1: $HALT_{TM}$ is undecidable.

Proof: Suppose $HALT_{TM}$ is decidable. Let R be the TM that decides $HALT_{TM}$. We will use R to build a decider for A_{TM}.

A_{TM} decider
- Given $\langle M, w \rangle$ – A_{TM} instance
- Run $\langle M, w \rangle$ in R. If R rejects, then reject
- If R accepts, then simulate M on w.
    - Accept if M accepts
    - Reject if M rejects

M does not halt on w

– M accepts w : R accepts $\langle M, w \rangle$.

Anyway, the point the statement that I want to say here is that HALT_TM is undecidable. So, we will see two approaches or we will see two ways to show this. They are essentially the same thing, but then we will present it in two different ways. So, there is a small thing that we are doing slightly differently from the textbook, Sipser textbook. The Sipser textbook illustrates the notion of reductions using a lot of examples and a lot of things, and towards the end of the chapter Sipser defines reductions. However, we have already defined reductions in the previous lecture, and now we are going to go through the examples.

Anyway first, so here we will show that HALT_TM is undecidable. So, first will assume it is decidable and show that this implies that an A_TM is decidable. And since this contradicts the known fact that A_TM is undecidable, this should imply that HALT_TM is undecidable. This is the first approach, so, let us see what the approach is. So, suppose there is a decider for $HALT_{TM}$, let us call it R. So, let R be the Turing machine that decides HALT_TM, so, this is decider for HALT_TM, let us call it R. So, now let us see how we can use R to build a decider for A_TM.

So, suppose we are given an instance of A_TM M comma w, we have to determine whether M accepts w or not. We want a decision, decider for this. So, what we do is first we run you, we feed M and w to R. So, R is a decider for HALT_TM, so, this will tell R is a decider for $HALT_{TM}$. So, R should tell whether M comma w is a YES instance or a NO instance of HALT_TM. In other words, R should tell whether M halts on w, yes or no? Suppose, R rejects, then we reject. And so, what does it mean? If R rejects it, it means that it is a no instance of $HALT_{TM}$. In other words, M does not halt on w, so, M does not halt on w is, then we reject. So, maybe I will just note it here, M does not halt on the w that is when we reject here.

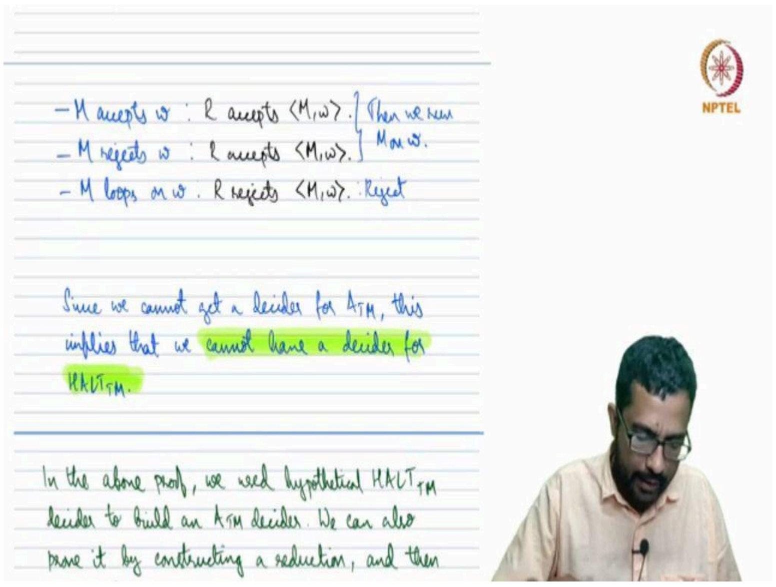
Suppose, R accepts the pair M comma w. What does it mean? This means that it is a YES instance of HALT_TM, meaning M halts on w. But, then halting could mean two possibilities further, M could accept or M could reject. So, now it is easy because now we are assured that M halts on W, so, then we just run M on w, we just simulate M on w.

Now, we just wait for the response because we know that it halts. So, we just complete the computation and if it accepts, we say yes and we accept. If it rejects, we say reject, so that is the decider for the A_TM. So, given M comma w, basically we want to run M on w, this is what we discussed earlier in the last week. We can run M on w and output whatever we get.

But, the issue that we ran into was that what if M loops on W? That was the issue. So, now that issue here is taken care of by R, so, R is a decider for HALT_TM. So, we ask R whether M halts on W, and if R rejects, we know that M does not halt on W. And then if M does not halt, it does not accept, so, then we can reject the instance.

So, we know that M does not accept w and then we can reject, even in A_TM, we can reject. If it, if R accepts M comma w, which means M halts on w, so now we know M halts. So, now we simply run M on w, so we accept if M accepts and reject if M rejects. So, I have just written some table in the same thing that I have been explaining.

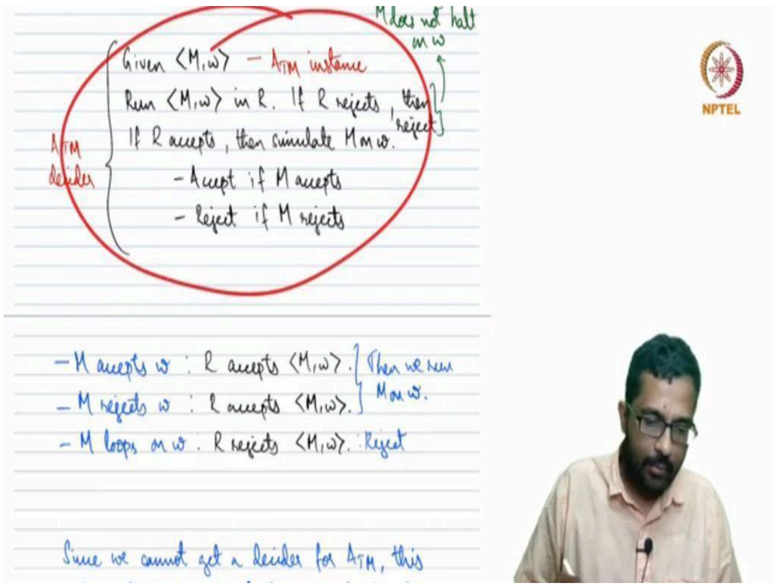
(Refer Slide Time: 07:30)



- M accepts w : R accepts $\langle M,w \rangle$. } Then we run M on w.
- M rejects w : R accepts $\langle M,w \rangle$. }
- M loops on w . R rejects $\langle M,w \rangle$. : Reject

Since we cannot get a decider for A_{TM}, this implies that we cannot have a decider for $HALT_{TM}$.

In the above proof, we used hypothetical $HALT_{TM}$ decider to build an A_{TM} decider. We can also prove it by constructing a reduction, and then



M does not halt on w

A_{TM} decider

Given $\langle M,w \rangle$ — A_{TM} instance

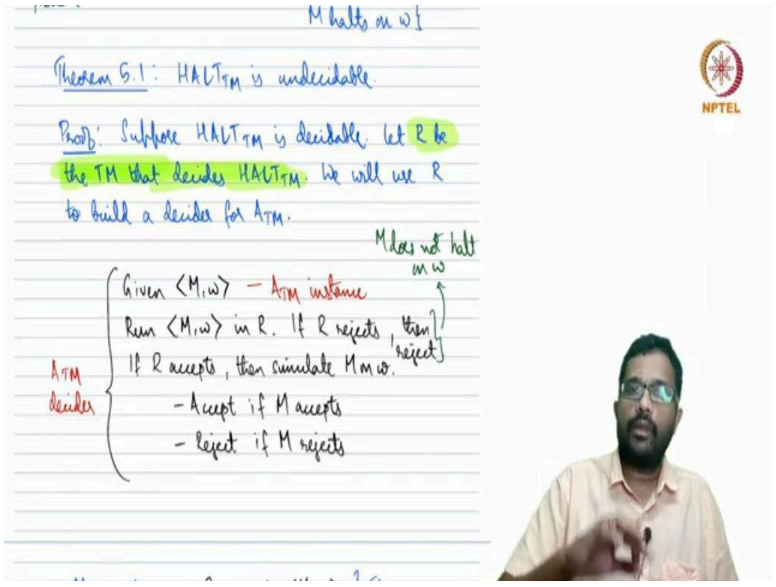
Run $\langle M,w \rangle$ in R. If R rejects, then reject

If R accepts, then simulate M on w.

- Accept if M accepts
- Reject if M rejects

- M accepts w : R accepts $\langle M,w \rangle$. } Then we run M on w.
- M rejects w : R accepts $\langle M,w \rangle$. }
- M loops on w . R rejects $\langle M,w \rangle$. : Reject

Since we cannot get a decider for A_{TM}, this



M halts on w}

Theorem 5.1: $HALT_{TM}$ is undecidable.

Proof: Suppose $HALT_{TM}$ is decidable. Let R be the TM that decides $HALT_{TM}$. We will use R to build a decider for A_{TM}.

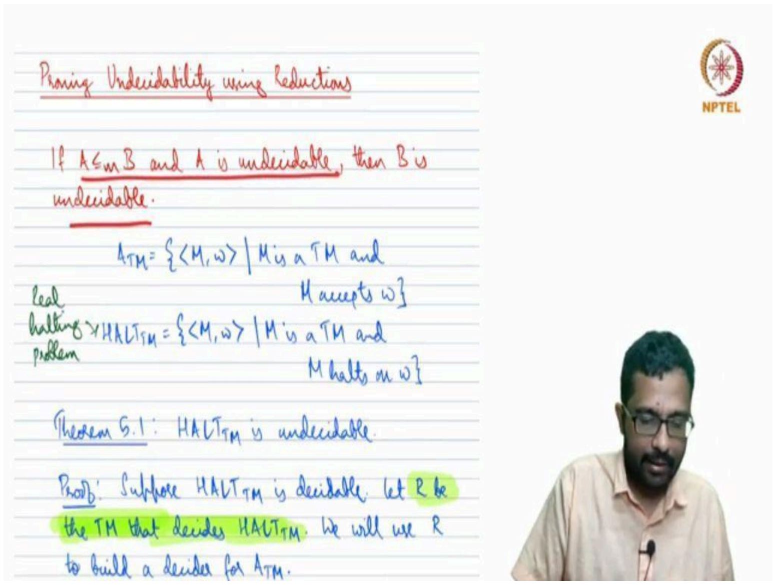
A_{TM} decider

Given $\langle M, w\rangle$ – A_{TM} instance

Run $\langle M, w\rangle$ in R. If R rejects, then reject. ← M does not halt on w

If R accepts, then simulate M on w.

- Accept if M accepts
- Reject if M rejects



Proving Undecidability using Reductions

If $A \leq_m B$ and A is undecidable, then B is undecidable.

$A_{TM} = \{\langle M, w\rangle \mid M$ is a TM and M accepts w$\}$

Real halting problem → $HALT_{TM} = \{\langle M, w\rangle \mid M$ is a TM and M halts on w$\}$

Theorem 5.1: $HALT_{TM}$ is undecidable.

Proof: Suppose $HALT_{TM}$ is decidable. Let R be the TM that decides $HALT_{TM}$. We will use R to build a decider for A_{TM}.

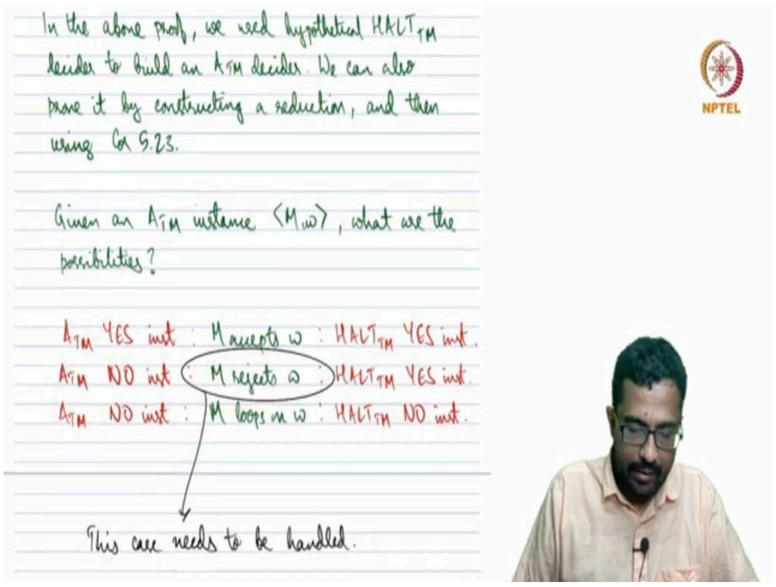
If M accepts w, then R accepts M, w, R accepts M, w. And then we can say then, then we run, sorry, then we run M on w. If M rejects, if M rejects w, even then it is halting on w. So, in that case also the decider for HALT_TM R should accept M comma w. Again, we run R sorry, again we run M on w in that case as well. The only issue is when M loops on w, so then we want to, and then we want to reject the pair M comma w. So, now this is made easy by the decider for HALT_TM that is R, so we run R on M, w first, so R rejects. So, then we reject, so then in this case also we reject.

So, these are the three cases: M accepts, rejects and loops. In each of these three cases, we are able to determine what, determine what happens and then make a decision accordingly. This decider, the decider that we just built over here, this never loops. Because if M loops on w, we never run M on w, in that case, we know because of R. So, now the only main component of this A_TM decider is the HALT_TM decider. And by assumption, the HALT_TM decider exists. So, because we know that A_TM is undecidable, in other words, A_TM decider does not exist, this should mean that HALT_TM decider, it should also not exist. Hence, HALT_TM is undecidable.

So, we cannot have a HALT_TM decider, hence it is undecidable, so that is the proof. So, what we did here is to just build a decider for A_TM from HALT_TM. So, this is not really like a reduction in the way we defined it in the previous lecture. A reduction was a mapping and we from one instance we constructed, from A instance we considered, B instance et-cetera. So, here we are not doing that. We are just using HALT_TM to build a decider for A_{TM}. So, now we will see the same… same proof, slightly differently, slightly different perspective. What we will do is we will reduce HALT, sorry we will reduce A_TM to HALT_TM.

So, we said that if A reduces to B and A is undecidable, then B is undecidable. So, what we will do is we will reduce A_TM to HALT_TM. Since, we know that A_TM is undecidable, this implies HALT_TM is undecidable. So, the ideas the idea of this reduction I have already said it in this lecture as well as one of the previous lectures, so, we need to reduce A_TM to HALT_TM. So, let us see what are the possiblities.

(Refer Slide Time: 11:02)


In the above proof, we used hypothetical $HALT_{TM}$ decider to build an A_{TM} decider. We can also prove it by constructing a reduction, and then using Cor 5.23.

Given an A_{TM} instance $\langle M, w \rangle$, what are the possibilities?

A_{TM} YES inst : M accepts w : $HALT_{TM}$ YES inst.
A_{TM} NO inst : M rejects w : $HALT_{TM}$ YES inst.
A_{TM} NO inst : M loops on w : $HALT_{TM}$ NO inst.

This case needs to be handled.

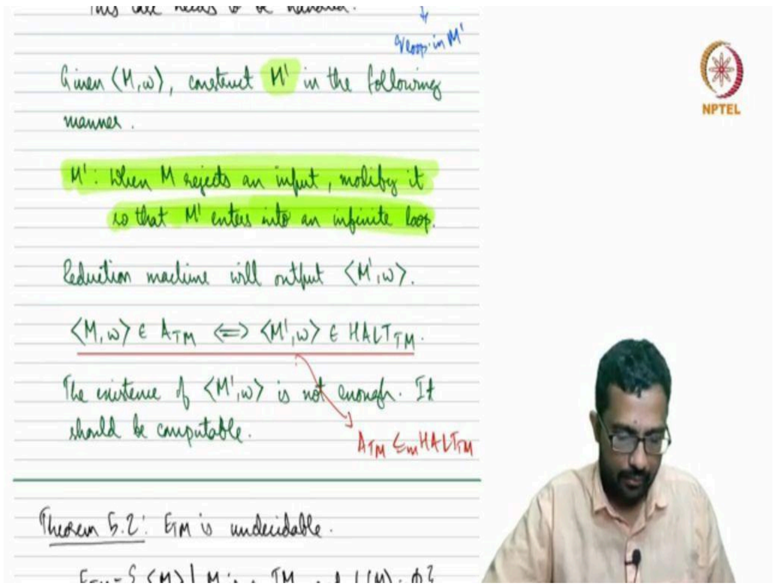
So, A_TM let us say the instance is M, w. So, A_TM, the instance is M, w. And now what are the three possibilities? Either M accepts w or M rejects w, or M loops on w. So, M accepts w is the first instance that is written here M accepts w, it is a YES instance for A_TM, because

M accepts w. It is also a YES instance for HALT_TM, if M, w was HALT_TM, it is an YES instance. If M rejects w, it is a NO instance for A_TM, but the YES instance for HALT_TM. Because HALT is just a HALT_TM is just checking whether it halts or not, so rejection is also HALT. So, it is a YES instance for HALT_TM, but NO instance for A_TM.

If M loops on w, it never halts, never accepts, never rejects, it halts, sorry it loops. If it loops on w, then it is a NO instance for A_TM, because M does not accept. It is also a NO instance for HALT_TM, because it does not halt. So, these are the three possibilities. So, M, w as an instance, if you feed it to A_TM and HALT_TM, the outputs agree in the first case and the third case. The issue is with the second case. So, if the same instance worked when we need not do anything there, just the same M comma w could have been a reduction. But, then we need to do something about the second case, this needs to be handled, because it is NO instance for A_TM, but it is a YES instance for HALT_TM.

If we could do something, we could do some modification to the instance. So, whenever M rejects w, we do not halt, it becomes a NO instance for HALT_TM also, then it is then it will be fine. So, the question is can we modify the A_TM instance M, w? So, that whenever M rejects w, it does not halt. Or, we modify it into something, so that the modified instance never halts. So, when M rejects w, we want to construct some M' W', such that whenever M rejects w, M' loops on W'. So, in fact, what we will do is we will not construct a separate W', we will have the same W. But, we will construct it different M.

(Refer Slide Time: 13:32)



q_{loop} in M'

Given $\langle M, w\rangle$, construct M' in the following manner.

M': When M rejects an input, modify it so that M' enters into an infinite loop.

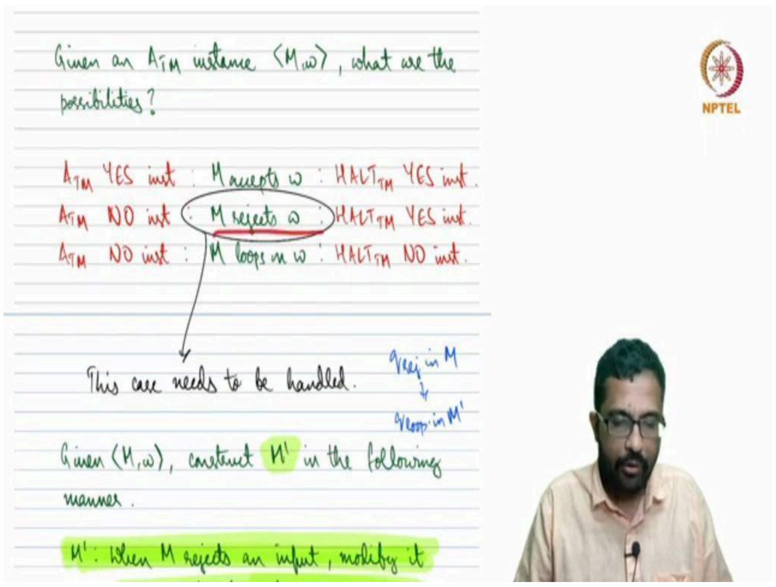
Reduction machine will output $\langle M', w\rangle$.

$\langle M, w\rangle \in A_{TM} \iff \langle M', w\rangle \in HALT_{TM}$.

The existence of $\langle M', w\rangle$ is not enough. It should be computable.

$A_{TM} \leq_m HALT_{TM}$

Theorem 5.2: E_{TM} is undecidable.



Given an A_{TM} instance $\langle M, w\rangle$, what are the possibilities?

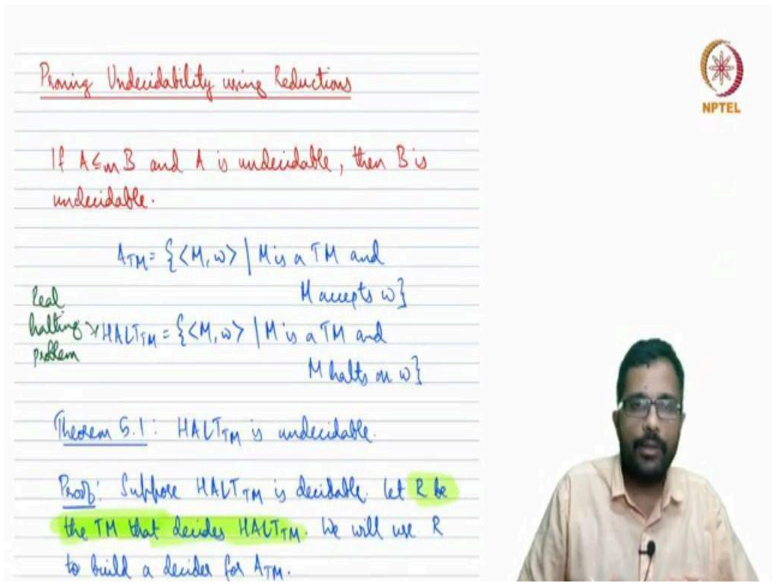
A_{TM} YES inst : M accepts w : $HALT_{TM}$ YES inst.
A_{TM} NO inst : M rejects w : $HALT_{TM}$ YES inst.
A_{TM} NO inst : M loops on w : $HALT_{TM}$ NO inst.

This case needs to be handled.

q_{rej} in M
$\downarrow$
q_{loop} in M'

Given $\langle M, w\rangle$, construct M' in the following manner.

M': When M rejects an input, modify it



Proving Undecidability using Reductions

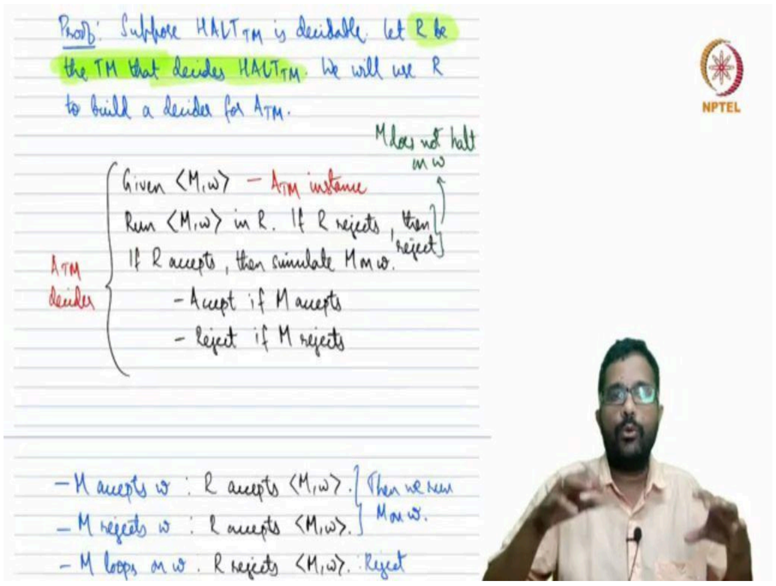
If $A \leq_m B$ and A is undecidable, then B is undecidable.

$A_{TM} = \{\langle M, w\rangle \mid M$ is a TM and M accepts $w\}$

Real halting problem $\rightarrow HALT_{TM} = \{\langle M, w\rangle \mid M$ is a TM and M halts on $w\}$

Theorem 5.1: $HALT_{TM}$ is undecidable.

Proof: Suppose $HALT_{TM}$ is decidable. Let R be the TM that decides $HALT_{TM}$. We will use R to build a decider for A_{TM}.



Proof: Suppose $HALT_{TM}$ is decidable. Let R be the TM that decides $HALT_{TM}$. We will use R to build a decider for A_{TM}.

M does not halt on w

A_{TM} decider
- Given $\langle M, w\rangle$ — A_{TM} instance
- Run $\langle M, w\rangle$ in R. If R rejects, then reject
- If R accepts, then simulate M on w.
  - Accept if M accepts
  - Reject if M rejects

- M accepts w : R accepts $\langle M, w\rangle$. } Then we run M on w.
- M rejects w : R accepts $\langle M, w\rangle$. }
- M loops on w : R rejects $\langle M, w\rangle$. Reject

So, we will construct M'. So, what M' will do is it will follow M mostly. But, instead whenever M has to reject, M' goes into a loop. So, what we can this is not that difficult to accomplish, I will not write in detail. But so, so q rejects in m, we replace by some looping state, Loop q loop in M'. So, q loop can be a loop state, meaning once you reach q loop, you just get stuck there. The transitions just keeps you in q loop and never go out. So, if we make this modification to M, whenever M was supposed to reject a string, M' will now just start looping.

So, that is what we want and that is what I have written here. M' this is the main idea of M'. Whenever M rejects an input, M', we want to enter into an infinite loop. So, basically, we whenever M rejects, that was the issue here, this second line here. M whenever M rejects w, it is a NO instance for A_TM. The HALT_TM instance we also wanted to know instance for

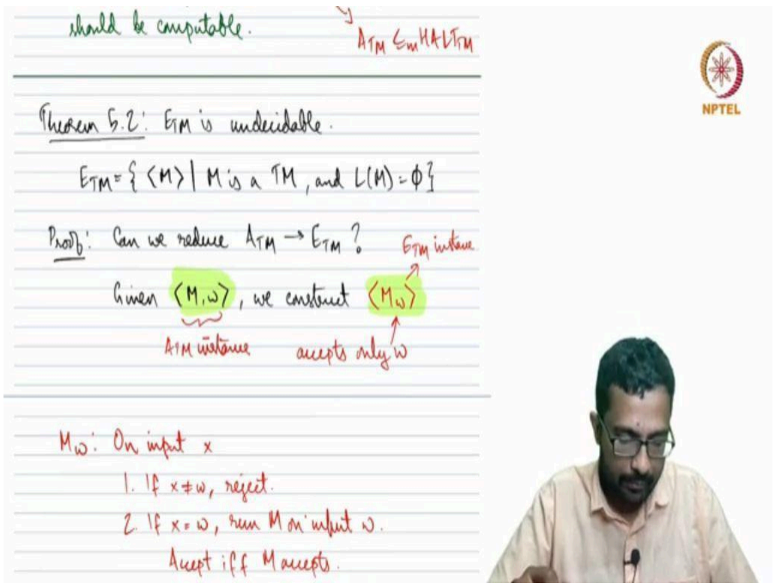
HALT_TM. So, we make it not halt, we make it loop. So, the reduction machine will just output M' and w, so we know how to construct M'. It is just a simple modification. When you replace q reject in M with q loop in M', so where it just loops, so that is a reduction process.

So, the construction of M', so easy, so this is a very small modification we can easily do it. And it is not that difficult to see now. So, now, the NO instance of A_TM corresponds to a NO instance of HALT_TM also, because the M comma w is a NO instance of A_TM, M rejects w. The map instances M' w. So, M' loops on w whenever M rejects w. Whenever M accepts w, M' also accepts w, whenever M loops on w also, M' loops on w. So, that there is no change in the first and second, the first and last situations, the difference only in the, is only in the second case when M rejects an output, and that is now hand built.

So, this means that whenever M comma w YES instance of A_TM, if and only if M' comma w is a YES instance of HALT_TM. So, this shows that just to write, so, maybe use a different color, red color, this indicates sorry, this indicates that A_TM reduces to HALT_TM. This means I do not think I wrote it anyway, this means that HALT_TM is undecidable. That is what we said at the beginning. If A reduces to B, A is undecidable, then B is undecidable.

So, that is how we show that HALT_TM is undecidable. So, first we saw the sort of direct approach, where we use the A_TM, sorry, the HALT_TM decider to construct the A_TM decider. Now, here we are actually presenting a reduction from A_TM to HALT_TM, where we modify the A_TM instance to construct a HALT_TM instance, so that it is a proper reduction. YES, instance map to YES instance, and NO instance maps to NO instance. Now, some more languages using Turing machines, which we will show to be undecidable.

(Refer Slide Time: 17:27)



should be computable. $A_{TM} \leq_m HALT_{TM}$

Theorem 5.2: E_{TM} is undecidable.

$E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM, and } L(M) = \phi \}$

Proof: Can we reduce $A_{TM} \rightarrow E_{TM}$?

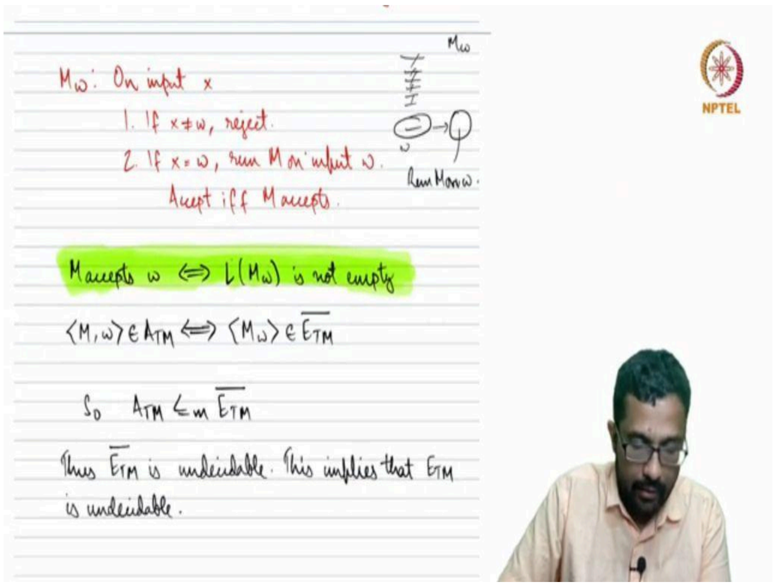
Given $\underbrace{\langle M, w \rangle}_{A_{TM} \text{ instance}}$, we construct $\langle M_w \rangle$ E_{TM} instance accepts only w

```
M_w: On input x
        1. If $x \neq w$, reject.
        2. If $x = w$, run M on input w.
            Accept iff M accepts.
```

So, the language is first language is E_TM. E_TM given a TM, we are asking whether this accepts this recognizes the empty language. Given a TM M, we are asking whether this recognizes the empty language? So, this is also undecidable, so let us see why. So, we will reduce A_TM to E_TM here. So, given an A_TM instance, which is M, w, we will construct an E_TM instance <M,w>, sorry <M_w>, so this A_TM instance. A_TM instance will construct E_TM instance.

So, <M,w> and<M_w>, means it is a Turing machine alone, there is no string because E_TM instance has only one Turing machine. But, it we will use will use w will be important in the construction of M, <M_w>. So, what is <M_w> do? So, whenever M accepts w, we want <M_w> to be. So, in fact we will see what the reduction is and then I will talk about this, so, <M_w> is this.

(Refer Slide Time: 18:57)



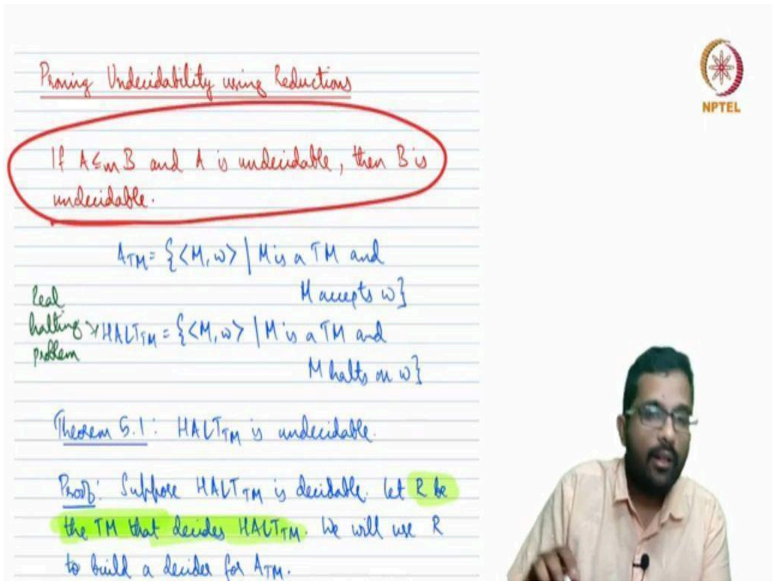
M_w: On input x
1. If $x \neq w$, reject.
2. If $x = w$, run M on input w.
Accept iff M accepts.

M accepts w $\iff$ $L(M_w)$ is not empty

$\langle M, w\rangle \in A_{TM} \iff \langle M_w\rangle \in \overline{E_{TM}}$

So $A_{TM} \leq_m \overline{E_{TM}}$

Thus $\overline{E_{TM}}$ is undecidable. This implies that E_{TM} is undecidable.



Proving Undecidability using Reductions

If $A \leq_m B$ and A is undecidable, then B is undecidable.

$A_{TM} = \{\langle M, w\rangle \mid$ M is a TM and M accepts w$\}$

Real halting problem $\rightarrow$ $HALT_{TM} = \{\langle M, w\rangle \mid$ M is a TM and M halts on w$\}$

Theorem 5.1: $HALT_{TM}$ is undecidable.

Proof: Suppose $HALT_{TM}$ is decidable. Let R be the TM that decides $HALT_{TM}$. We will use R to build a decider for A_{TM}.

Wherever there is an input x to <M_w>, so x is the input to <M_w>, x is the input to <M_w>. If x, so now M, w knows w, so, w is like a fixed string that it is aware of. If x is the same, if x is not w, it rejects it. So, it first checks whether x is equal to w, if x is not equal to w, it rejects. So, that is the first step that M does, sorry <M_w> does. So, it rejects for any input that is not w, so, the only string that it may accept is w.

So, let us see what it does when the input is w. If the input is w, it runs M on w, and then does whatever M does. So, M may not be a decider, M', M, w may not be also, may also not be a decider. So, if the input to<M_w> is not w, it rejects. If the input to <M_w> is w, we run M on the input w, and we do whatever M does. If M accepts we accept, we meaning <M_w>. If M accepts, <M_w> accepts. If M rejects, <M_w> rejects, if it loops, <M_w> also loops. So, of course, these loops mean it is just, it just loops there is no reporting. So, let us see what is happening here.

Suppose, M accepts so there are many inputs, there are many possible inputs that many strings are possible for as inputs for <M_w>. So, it rejects all of these inputs, the only string that it even considers is a string w. If it is if the input is w, then it checks whether M accepts w, or does it run M on w? Run M on w and thus do whatever M does.

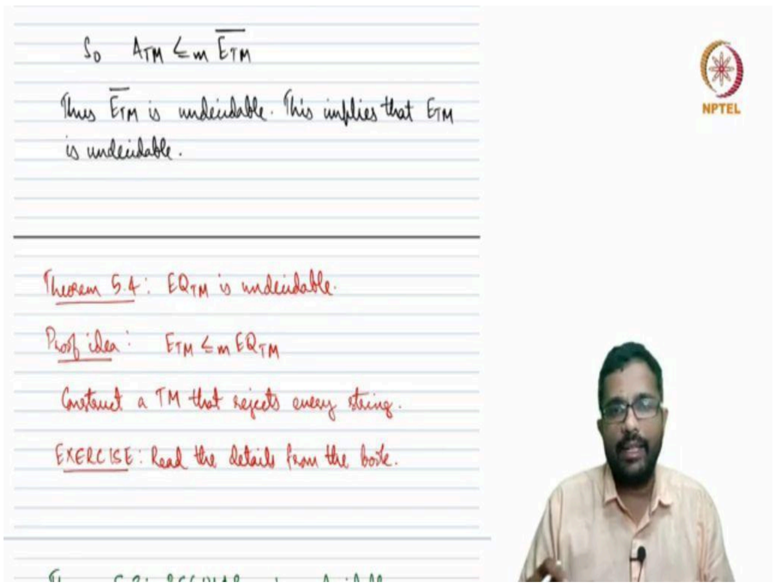
So, it rejects everything that is not w, and then runs M on w if the input is w. So, if M accepts w, then <M_w> also accepts w. If M accepts w, then <M_w> accepts w. If M rejects w, then <M_w> rejects everything that is not w, that is always the case. If <M_w> sorry, if M rejects w, then <M_w> also rejects w, which means <M_w> does not accept anything.

If M loops on w, even then <M_w> does not accept w, even then <M_w> does not accept any string, any string in the alphabet over the alphabet. Because all the strings that are not w are immediately rejected, the only string then it has any chance of accepting is w. So, if M rejects or loops on w, then <M_w> also does not accept w. Which means that if M does not accept w, <M_w> does not accept any string, so, the only possibility that <M_w> accept some string is if M accepts w. So, that is what I have written here. M accepts w means the language recognized by L <M_w> is not empty. So, the language recognized by <M_w> is just, there are only two possibilities, either it is empty or it is a single string w.

If M accepts w, L(M_w) is the single string w. If M does not accept w, L(Mw) is empty. M accepts w if and only if L (Mw) is not empty. In other words, we are saying that M accepts w means Mw is in A_TM, and L Mw not empty means that <M_w> is in the complement of A_TM. It is not an empty language. So, what we have here is a reduction from A_TM to the complement of E_TM. So, E_TM asks whether the given Turing machine recognizes the empty language. This E_TM complement means it is the given Turing machine that recognizes something which is non-empty. So, A_TM reduces to E_TM complement, this means that E_TM complement is undecidable.

This means that E_TM complement is undecidable by our theorem that I stated at the beginning of the lecture, so, this means that E_TM complement is undecidable. And if some language L complement is undecidable, then L is also undecidable, because if I had a decider for L, I could just run the decider for L, and flip the accept-reject states to get a decider for L complement. So, this implies that E_TM is also undecidable. So, the trick here was to create a Turing machine that only accepts w, so, if and only possibly accepts w. So, whenever M rejects or loops on w, <M_w> does not accept anything and the language is empty. Whenever M accepts w, the language is not empty, so, that is the proof here.

(Refer Slide Time: 25:17)



So $A_{TM} \le_m \overline{E_{TM}}$

Thus $\overline{E_{TM}}$ is undecidable. This implies that E_{TM} is undecidable.

Theorem 5.4: EQ_{TM} is undecidable.

Proof idea: $E_{TM} \le_m EQ_{TM}$

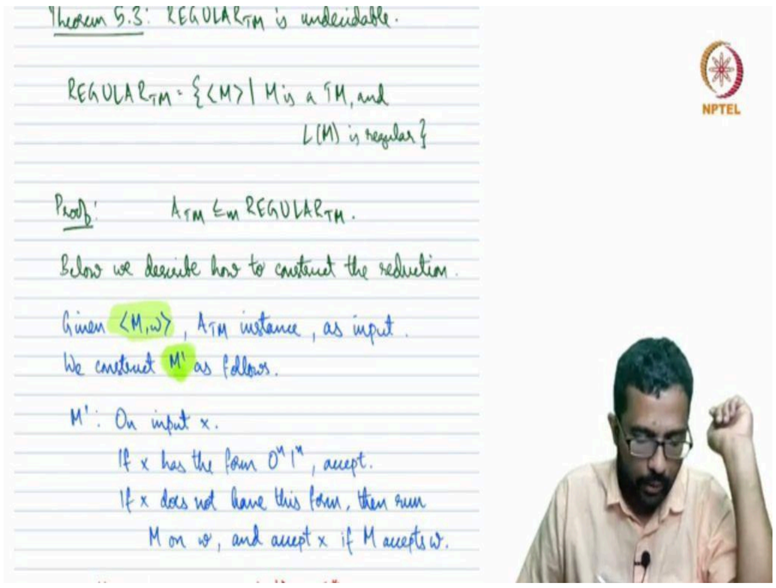
Construct a TM that rejects every string.

EXERCISE: Read the details from the book.

There is one other language EQ $_{TM}$. So, EQ $_{TM}$ is given to Turing machines we are asking whether they recognize the same language, just like EQ, EQ-DFA or EQ-CFG. This is also undecidable. There is a very straightforward reduction from E_TM to EQ $_{TM}$. So basically, we constructed a Turing machine that rejects every string, we constructed a Turing machine that rejects every string.

So, to determine whether a given Turing machine recognizes the empty language, we can ask whether this Turing machine is equivalent to the constructed Turing machine, which also recognizes the empty language. So, I do not want to get into the details. It is very straightforward and it’s all the similar lines to other proofs that we have seen, so I suggest to read the proof from the book.

(Refer Slide Time: 26:17)



Theorem 5.3: REGULAR_{TM} is undecidable.

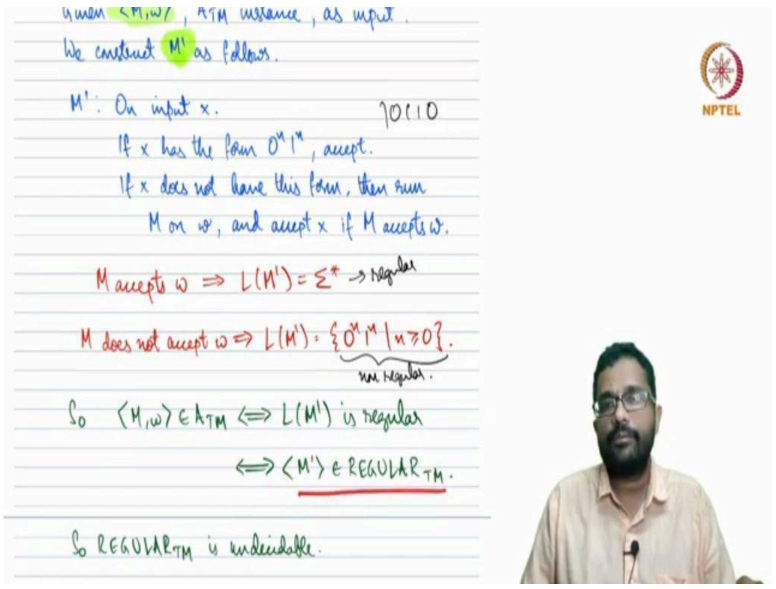
$\text{REGULAR}_{TM} = \{\langle M\rangle \mid M$ is a TM, and $L(M)$ is regular$\}$

Proof: $A_{TM} \leq_m \text{REGULAR}_{TM}$.

Below we describe how to construct the reduction.

Given $\langle M, w\rangle$, A_{TM} instance, as input.
We construct M' as follows.

M': On input x.
If x has the form $0^n 1^n$, accept.
If x does not have this form, then run M on w, and accept x if M accepts w.



Given $\langle M, w\rangle$, A_{TM} instance, as input.
We construct M' as follows.

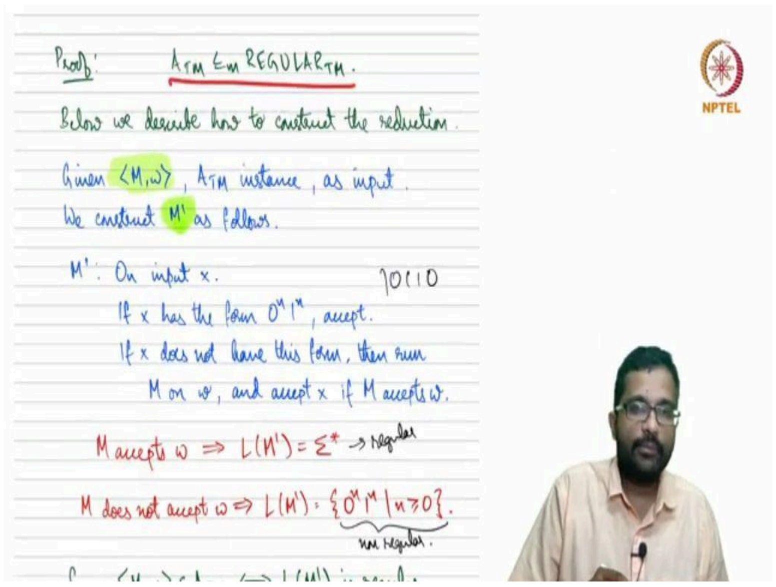
M': On input x. 10110
If x has the form $0^n 1^n$, accept.
If x does not have this form, then run M on w, and accept x if M accepts w.

M accepts $w \Rightarrow L(M') = \Sigma^* \rightarrow$ regular

M does not accept $w \Rightarrow L(M') = \underbrace{\{0^n 1^n \mid n \geq 0\}}_{\text{non regular}}$.

So $\langle M, w\rangle \in A_{TM} \Longleftrightarrow L(M')$ is regular
$\Longleftrightarrow \langle M'\rangle \in \text{REGULAR}_{TM}$.

So REGULAR_{TM} is undecidable.



Proof: $A_{TM} \leq_m REGULAR_{TM}$.

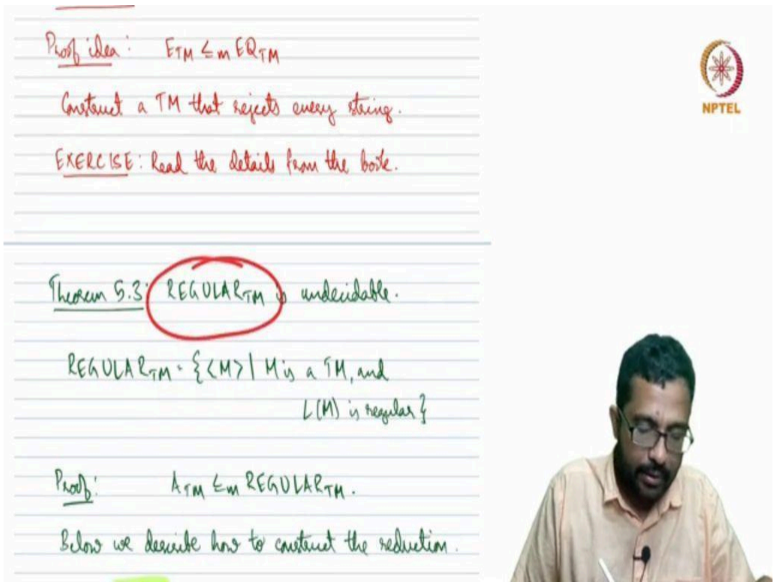
Below we describe how to construct the reduction.

Given $\langle M, w\rangle$, A_{TM} instance, as input.
We construct M' as follows.

```
M': On input x.                                   10110
      If x has the form $0^n 1^n$, accept.
      If x does not have this form, then run
          M on w, and accept x if M accepts w.
```

M accepts $w \Rightarrow L(M') = \Sigma^* \rightarrow$ regular

M does not accept $w \Rightarrow L(M') = \underbrace{\{0^n 1^n \mid n \geq 0\}}_{\text{non regular}}$.



Proof idea: $E_{TM} \leq_m EQ_{TM}$

Construct a TM that rejects every string.

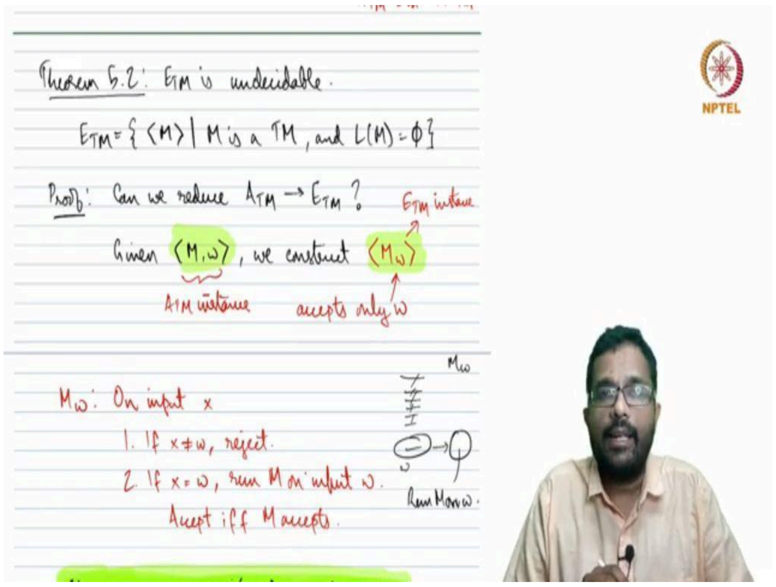
EXERCISE: Read the details from the book.

Theorem 5.3: $REGULAR_{TM}$ is undecidable.

$REGULAR_{TM} = \{\langle M\rangle \mid M \text{ is a TM, and } L(M) \text{ is regular}\}$

Proof: $A_{TM} \leq_m REGULAR_{TM}$.

Below we describe how to construct the reduction.



Theorem 5.2: E_{TM} is undecidable.

$E_{TM} = \{\langle M\rangle \mid M \text{ is a TM, and } L(M) = \phi\}$

Proof: Can we reduce $A_{TM} \rightarrow E_{TM}$?

Given $\underbrace{\langle M, w\rangle}_{A_{TM}\text{ instance}}$, we construct $\langle M_w\rangle$

E_{TM} instance

accepts only w

```
M_w: On input x
        1. If x ≠ w, reject.
        2. If x = w, run M on input w.
           Accept iff M accepts.
```

The next language is REGULAR $_{TM}$. So, what is REGULAR $_{TM}$? REGULAR $_{TM}$ is given a Turing machine. We are asking whether the language recognized by this Turing machine is a regular language. Given a Turing machine, we are asking if the language recognized by this Turing machine is a regular language, so this is also undecidable. Again, we show it by reducing it from A_TM, so we reduce A_TM to REGULAR $_{TM}$. And this implies that REGULAR $_{TM}$ is undecidable, so let us see how to construct the reduction. So, given an A_TM instance, will show how to construct a REGULAR $_{TM}$.

So, notice the REGULAR $_{TM}$ instance is just one description of a Turing machine. So, given an instance, <M,w> which is an A_TM instance, will construct a Turing machine M'. So, this will be the REGULAR $_{TM}$ instance, M' will be the REGULAR $_{TM}$ instance. Let us see what it does. M' when it gets an input x, so x is the input of M'. What it does is it first checks whether x has the form 0 power n, 1 power n, notice 0 power n, 1 power n where I want the number of 0s and the number of 1s to be the same. So, both have the same repetition, both have both should appear n times.

So, if the x if the input x has is of this form, we accept, nothing else needs to be checked. If it is not of this form, so suppose it is some other string, let say 0110 some something like that, or something 101 some other form. Then what we do is we run M on w, we run M on w, so, where M and w are as in the A_TM instance. So, M' actually depends on M and w, M' knows what is M, what knows what is w. Then, we run M on w. Again, it is not x but w, so, we run M on w. And if M accepts w, we accept x, if M accepts w, M' accepts x. Let me repeat what M' does.

Given the input x, it checks whether the input is of the form 0 power n, 1 power n, if it is of that form it accepts. If it is not of this form, it checks whether M accepts w by running M on w. It accepts x only when M accepts w, so, it runs M on w, and it accepts x if M accepts w. So, let us see both cases. Suppose M accepts w, suppose this always happens, suppose M accepts w. Now, all the strings that are of 0 power n, 1 power n form are accepted. If x does not have this form, we run M on w, and accept x is M accepts x, sorry, M accepts w. If M accepts w, we accept x.

So, if M accepts w, we accept x, which means that all x is accepted. If x is of this form, the 0-power n and 1 power n we anyway accept, otherwise we accept over here. But, in short all x's are accepted. In other words, the language recognized by the Turing machine M' is Σ^*, it accepts all the possible strings. If M does not accept w, if M does not accept, so, then here it does not accept w, which means, here we do not accept x. Which means if x does not have the requisite form, meaning if x is not of the form 0 power n, 1 power n, it is not accepted.

So, the only x that is going to be accepted are through this line. The only x that are going to be accepted are going to be through this line, 0 power n, 1 power n. So, if M does not accept w, the only x that is going to be accepted are 0 power n, 1 power n, which means the language recognized by the Turing machine M' is just the set of all strings of the form 0 power n, 1 power n. I will just repeat this because it is kind of the key part here. If x is the form 0 power n, 1 power n, you accept, if x is not of this form, we run M on w and accept x if M accepts w. If M accepts w, this means that strings both of the form 0 power n 1 power n, and not of the form are also accepted.

So, which means every string is accepted, so, the language recognized by M' is Σ*. If M does not accept w, then strings not of the form 0 power n, 1 power n are not accepted. So, the only strings accepted are 0 power n, 1 power n. So, but now, let us see what these languages are. If M accepts w, then the language Σ* is a regular language, Σ*, you can easily make it almost like a regular expression. If M does not accept w, then the language is one of our favourite non regular languages 0 power n 1 power n.

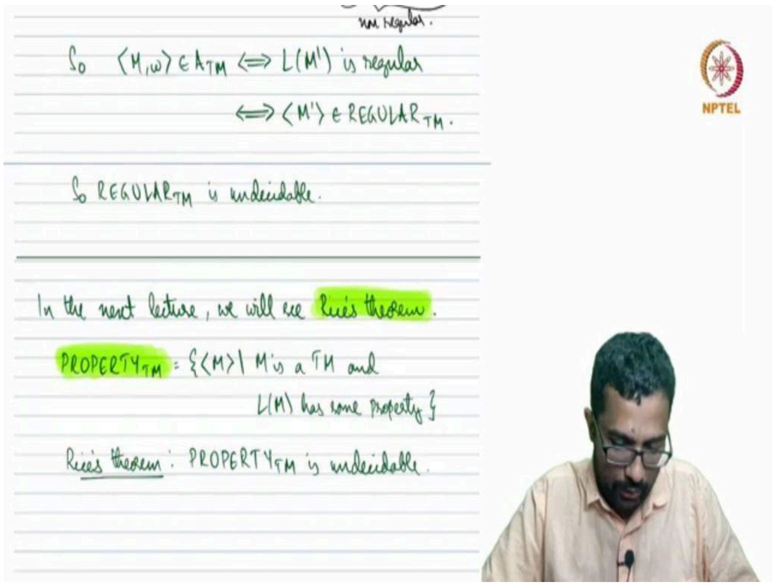
So, which means, M accepts w means that M' the language recognized by M' is regular, M does not accept w, means the language recognized by M' is not regular. So, the language recognized by M' is regular if and only if M accepts w. So, in other words, M comma w is an instance of A_TM, meaning if and only if L(M') is regular. And L(M') is regular, if and only if M' is in regular REGULAR $_{TM}$, by the definition of REGULAR $_{TM}$. So, M comma w is an A_TM if and only if M' is in REGULAR $_{TM}$, which means.

Now, we have a reduction from A_TM to REGULAR $_{TM}$, mapping reduction from A_TM to REGULAR $_{TM}$. And also, the construction of M' is something that is computable. So, M' we just have to encode these simple things. If x has of this form, we should accept, if it does not have this form, we know M and w, we just run M on w that is it. So, the construction of M' is computable. And also, we have this correspondence M comma w is in A_TM, if and only if M' is in REGULAR $_{TM}$. So, key thing again is that M accepts w if and only if L(M') is regular, that is the key thing.

So, M L, M accepts w which is this, if and only if L(M') is regular which is this. Hence, REGULAR $_{TM}$ is also undecidable. REGULAR $_{TM}$ is a Turing machine asking whether the language recognizes regular. So, I just want to quickly tell two things. So, REGULAR $_{TM}$ is asking given a Turing machine is the language recognized by the Turing machine regular. E_TM is asking given a Turing machine is the language recognized by the Turing machine, the empty language.

So, in both these cases, REGULAR $_{TM}$ as well as E_TM, we were given a Turing machine, and we were asked, is the language recognized by this Turing machine? Does the language by the Turing machine have this property? In one case, the property was that is this regular, the second sorry, and the second case this property was that is the language empty. So, both had that structure and we have showed that it is undecidable. So, in fact, it turns out that we can generalise this.

(Refer Slide Time: 35:15)



non-regular.

So $\langle M,w\rangle \in A_{TM} \iff L(M')$ is regular

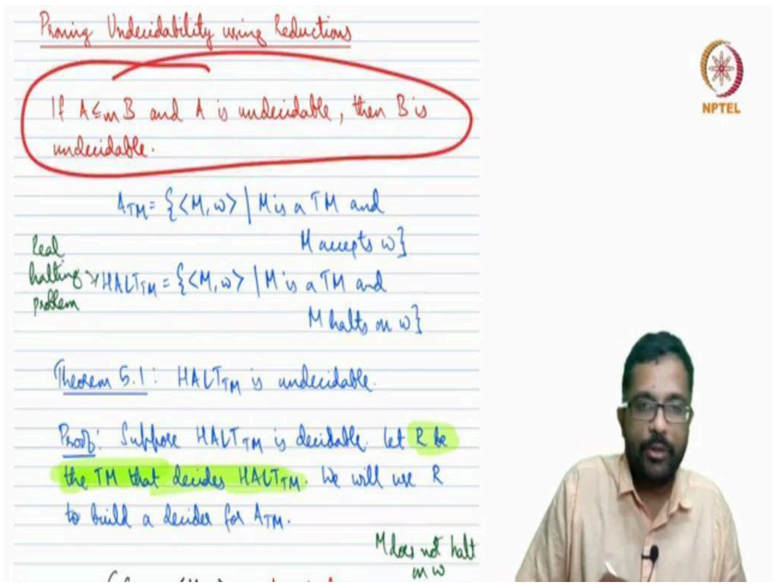
$\iff \langle M'\rangle \in REGULAR_{TM}$.

So $REGULAR_{TM}$ is undecidable.

In the next lecture, we will see Rice's theorem.

$PROPERTY_{TM} = \{\langle M\rangle \mid M$ is a TM and $L(M)$ has some property$\}$

Rice's theorem: $PROPERTY_{TM}$ is undecidable.



Proving Undecidability using Reductions

If $A \leq_m B$ and A is undecidable, then B is undecidable.

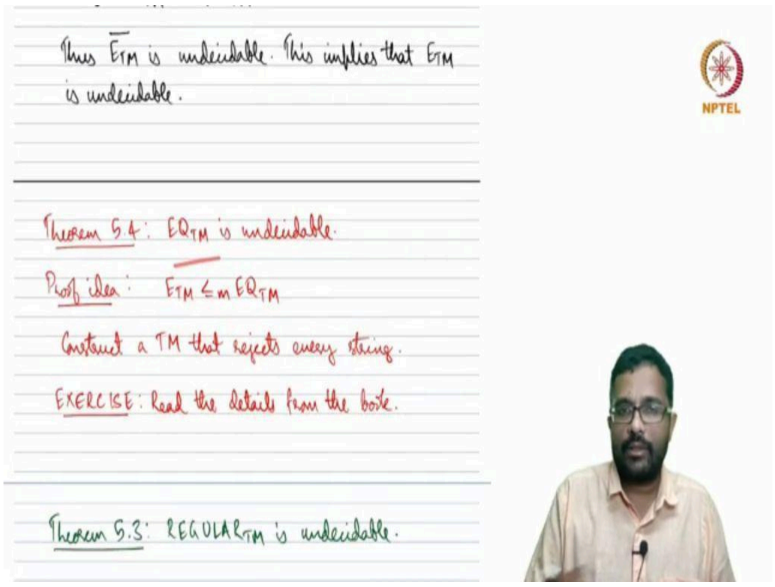
$A_{TM} = \{\langle M,w\rangle \mid M$ is a TM and M accepts $w\}$

Real halting problem $\rightarrow HALT_{TM} = \{\langle M,w\rangle \mid M$ is a TM and M halts on $w\}$

Theorem 5.1: $HALT_{TM}$ is undecidable.

Proof: Suppose $HALT_{TM}$ is decidable. Let R be the TM that decides $HALT_{TM}$. We will use R to build a decider for A_{TM}.

M does not halt on w


Thus $\overline{E_{TM}}$ is undecidable. This implies that E_{TM} is undecidable.

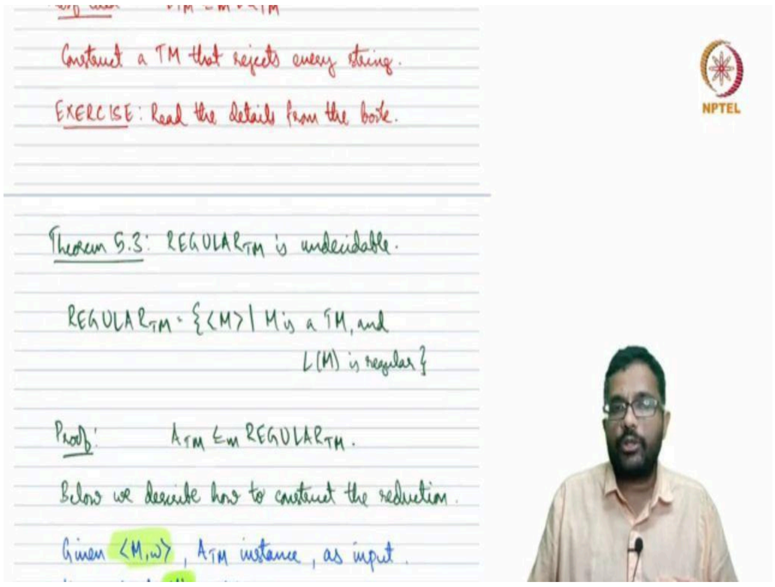
Theorem 5.4: EQ_{TM} is undecidable.

Proof idea: $E_{TM} \le_m EQ_{TM}$

Construct a TM that rejects every string.

EXERCISE: Read the details from the book.

Theorem 5.3: $REGULAR_{TM}$ is undecidable.


Construct a TM that rejects every string.

EXERCISE: Read the details from the book.

Theorem 5.3: $REGULAR_{TM}$ is undecidable.

$REGULAR_{TM} = \{\langle M \rangle \mid M \text{ is a TM, and } L(M) \text{ is regular}\}$

Proof: $A_{TM} \le_m REGULAR_{TM}$.

Below we describe how to construct the reduction.

Given $\langle M, w \rangle$, A_{TM} instance, as input.

So, the statement is called, this theorem is called Rice's theorem, so, the point is this. If so, consider the language called PROPERTY $_{TM}$. So, PROPERTY $_{TM}$ is like being given a Turing machine, we are asking whether the language recognized by the Turing machine is important, the language recognized by the Turing machine, does it have some property? We are not asking whether the Turing machine has a property, we are asking the language has some property? Like is the language regular? Is the language context-free? Does the language have at least one zero? Is the language all the strings in the language palindromes?

So, all these are questions of the language. So, we are asking, does the given Turing machines for the given Turing machine have the language recognized by this Turing machine? Does it satisfy some specific property? So, that is PROPERTY $_{TM}$. Rice's theorem says that whatever be the property or in, if the property is interesting or non-trivial, this question is undecidable.

So, we already saw that the REGULAR $_{TM}$ is undecidable, we already saw that E_TM is undecidable. So, the REGULAR $_{TM}$ is asking, does the given Turing machine recognize a regular language? E_TM is asking, does the given Turing machine recognize the empty language?

So, whatever may be the property that we're looking for, the testing of whether this Turing machine set recognizes a language with this property for any specific property, the answer is undecidable. It is not decidable. So, that we will see in the next lecture, and that is it for me in lecture number 40. We just use the statement that if A reduces to B and A is undecidable, to show that B is undecidable, we first showed HALT_TM is undecidable. Then, we showed E_TM is undecidable. Then, I briefly discussed EQ $_{TM}$ and asked you to read the proof, and finally, we saw REGULAR $_{TM}$ is undecidable.

And in the next lecture, lecture number 41, we will see Rice's theorem, where we will see that for a given Turing machine, testing whether this Turing machine, the language recognized by this Turing machine has a certain property. This is also undecidable for any property. So, see you in lecture 41. Thank you.