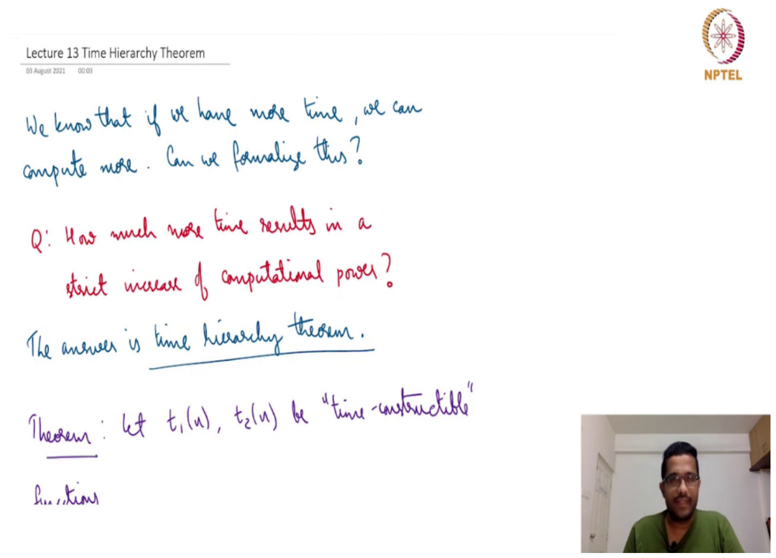
Computational Complexity
Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture -13
Time Hierarchy Theorem

(Refer Slide Time: 00:15)



Lecture 13 Time Hierarchy Theorem

03 August 2021 00:03

We know that if we have more time, we can compute more. Can we formalize this?

Q: How much more time results in a strict increase of computational power?

The answer is time hierarchy theorem.

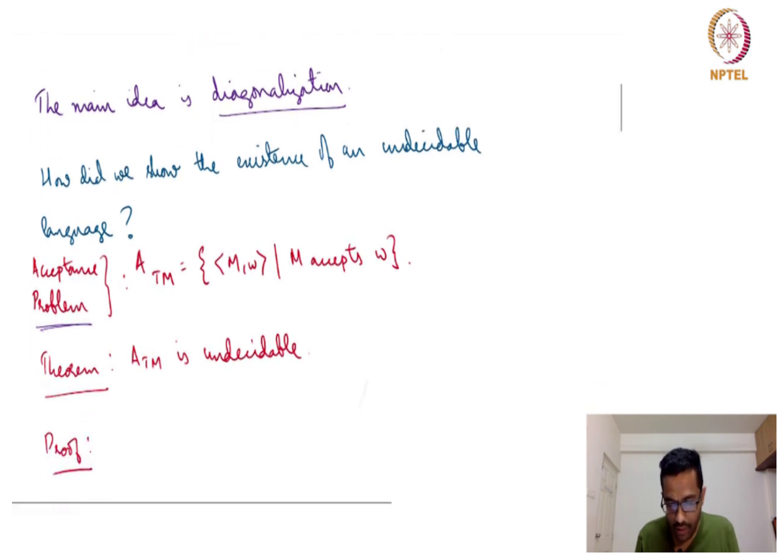
Theorem: Let $t_1(n)$, $t_2(n)$ be "time-constructible" functions

Hello and welcome to lecture 13 of computational complexity. In this lecture we will see time hierarchy theorem and this will be the last topic on the complexity based on complexity classes based on time at least for now. So, what is time hierarchy theorem? So, we know that if we give more time to a computer or a Turing machine, it can possibly do more. So, you allow it to compute for longer then maybe it can do better.

However, is there a way to formalize this? So, we know that non-determinism also seems to help we said that verification is easier than actually deciding but then we also know that we still do not know whether P is equal to NP or not. So, something seems to be true need not necessarily mean that it is, there is an easy way to show that it is true. But the statement that I just said about if we give a machine more time can it compute more?

This particular thing we will see that it is indeed the case. There are languages that we can compute if we have more time as compared to if we had lesser time. So, we will see what these languages are. So, more formally how much of, like, if you have more time how much more time should you need to give so, that there is we can compute more. So, we will see that this is called time hierarchy theorem.

(Refer Slide Time: 02:03)



The main idea is diagonalization.

How did we show the existence of an undecidable language?

Acceptance Problem } : $A_{TM} = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$.

Theorem : A_{TM} is undecidable.

Proof :

And I will just formally state it, let t_1 t_1 and t_2 t_2 also, called time constructable functions. So, we will at the end of the lecture I will define what is time constructable. So, it is a bit of a technical definition such that $t_1(n) \log t_1(n)$ $t_1(n) \log t_1(n)$ is $o(t_2(n))$ $o(t_2(n))$. Then the time $t_1(n)$ $t_1(n)$ is contained in the time $t_2(n)$ $t_2(n)$ this is what we will expect. So, what we are saying here is that $t_1(n)$ $t_1(n)$ has to be $o(t_2(n))$ $o(t_2(n))$ and not just $t_1(n)$ $t_1(n)$ must be $o(t_2(n))$ $o(t_2(n))$ we want $t_1(n) \log t_1(n)$ $t_1(n) \log t_1(n)$ to be $o(t_2(n))$ $o(t_2(n))$.

So, for instance let us say $t_1(n)$ $t_1(n)$ is n, and $t_2(n)$ $t_2(n)$ is n^2 n^2 then n log n is $o(n^2)$ $o(n^2)$. So, that is indeed the case of $t_1(n)$ $t_1(n)$ is n^2 n^2 and $t_2(n)$ $t_2(n)$ is n^3 n^3 even then $n^2 \log n$ $n^2 \log n$ is $o(n^3)$ $o(n^3)$. So, that is also fine. But so, you need at least this logarithmic separation. That is what we are saying here in that case is $DTIME(t_1(n))$ $DTIME(t_1(n))$ is contained in $DTIME\left(t_2(n)\right)$. $DTIME\left(t_2(n)\right)$.

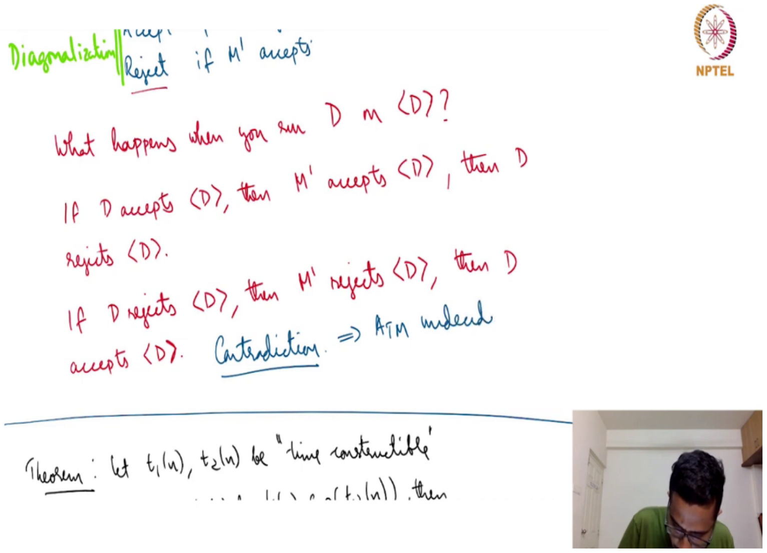
This is not very difficult to see because one is contained inside the other because $t_2(n)$ $t_2(n)$ is bigger than $t_1(n)$ $t_1(n)$. That is not so difficult to see. What time hierarchy theorem says that it is strictly contained inside the is $DTIME(t_1(n))$ $DTIME(t_1(n))$ is contained in $DTIME\left(t_2(n)\right)$ $DTIME\left(t_2(n)\right)$ and it is strict. Meaning there are some languages in $t_2(n)$ $t_2(n)$ ie., $DTIME(t_2(n))$ $DTIME(t_2(n))$ which are not there in is $DTIME(t_1(n))$ $DTIME(t_1(n))$. Meaning the languages that require time $t_2(n)$ $t_2(n)$ to compute but cannot be computed or cannot be decided in time $t_1(n)$ $t_1(n)$.

So, the main idea for this proof is by diagonalization. You may have seen diagonalization in the theory of computation course to show undecidability. However, we will do a very small recap here because the proof of time hierarchy theorem also is along very similar line. So, it will also help as a small warm up to see the diagonalization. So, what was the undecidable language you might have seen.

One of such language is what is called A_{TM} A_{TM}, the acceptance problem. So, given M a Turing machine, the encoding of a Turing machine and w which is an input, A_{TM} A_{TM} is a class of all M and w such that M accepts w. So, it is simply the question is to decide whether M accepts w. So, it seems like a very simple thing to do to decide whether M accepts w but it turns out to be extremely hard.

Because suppose the computation is ongoing you do not know how long to let it go. Either M could accept w, M could reject w or M may just not do either. It may just continue computation keep continuing computation. So, then how do you distinguish - is it just running on a loop or is it just going to stop at some point. How do you know when to stop. So, we do not know that and that is what makes this challenging.

(Refer Slide Time: 05:54)



Diagonalization

Reject if M' accepts

What happens when you run D on $\langle D \rangle$?

If D accepts $\langle D \rangle$, then M' accepts $\langle D \rangle$, then D rejects $\langle D \rangle$.

If D rejects $\langle D \rangle$, then M' rejects $\langle D \rangle$, then D accepts $\langle D \rangle$. Contradiction. $\Rightarrow A_{TM}$ undecid.

Theorem: let $t_1(n)$, $t_2(n)$ be "time constructible"

then

And this problem is undecidable given M and w whether M accepts w or not it is undecidable mainly because we do not know to determine whether it halts or not. So, sometimes this is also referred to as a halting problem. But there is also another language where M on w even that is undecidable but anyway let us stick to this now in order to not have any confusion. We will stick to A TM where we have to decide whether given turing machine M accepts a given string w.

So, now how do you show that it is undesirable. So, like many of these proofs go we start by contradiction. Suppose there exists $M' M'$ a turing machine that decides A TM. Suppose let us say it is decidable which means it has to have a decider and we call it $M' M'$. Now let us construct a turing machine D as follows. So, using $M' M'$. Now we will construct a turing machine D and what does D do given an input.

So, D just takes one input which is a description of a turing machine. So, now when I write it like this M with <M> it means I am giving D the input M which is the description of a turing machine m. So, the angular brackets denote it is a description of a turing machine. What do we do? We run $M' M'$ on this. So, what we do is we run $M' M'$. So, $M' M'$ is a decider for A TM which means it is given a string and a given a turing machine and a string and it starts to decide whether this turing machine accept this string.

So, run M' which is a known decider on this or an assumed decider on this. So, what is this? M' M' is a decider and these <M> and <M,<M>> are description of the turing machine M. Any Turing machine has a description. So, you can say list down the number of states you can list down the alphabet you can list down all the transitions like that you can write a description. Now then so, $M' M'$ is a decider which means it halts on all the inputs. So, then what do you do –you accept if M' rejects and you reject if $M' M'$ accepts.

So, it is very simple. So, basically what you do is you run the decider of A TM on M and its description, and you do the opposite. So, this is fine. So, far so, good. we have a decider for A_{TM} A_{TM} called $M' M'$ and we build another turing machine using a $M' M'$ as a subroutine which is D. So, where is the contradiction? Till now there is no contradiction.

Actually, the contradiction happens when you run when you run D on its own input. So, D is this turing machine that we just constructed and what happens when you run D on its own description. So, what will happen here, so, note that now D will take the place of m. So, now M prime will be run on D and its description. So, then $M' M'$ will see whether D accepts this description if D accepts this description then M prime will accept in which case there will be a reject over here.

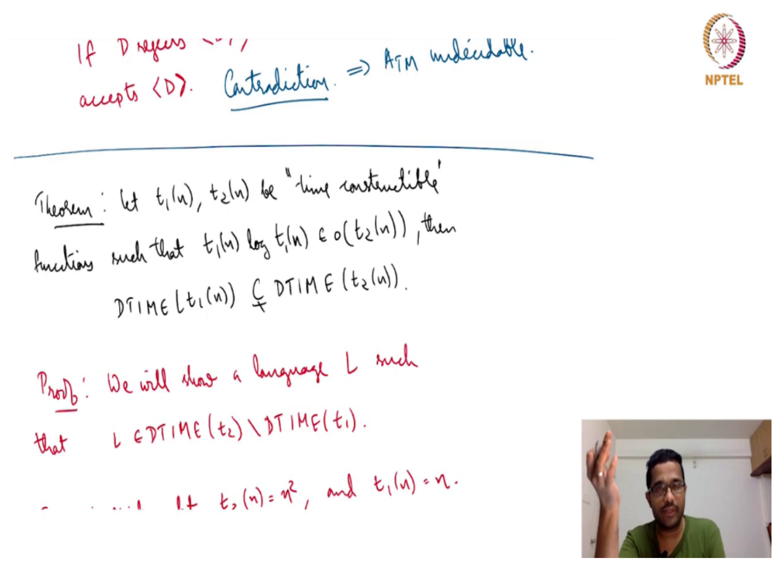
So, if D accepts its own description then $M' M'$ accepts D, its own the description of D then D ends up rejects it its own description. So, now note where we started and where we ended up. If D accepts its own description then $M' M'$ accepts the description of D but then D does the opposite of what $M' M'$ is doing. So, D will reject then the string. So, there is a contradiction what if the other way if D rejects its own description then $M' M'$ also rejects its own description but then D will flip the output, flip the result.

So, then D will accept its own description. So, it is a contradiction. This concept may be a bit confusing especially if you have not seen diagonalization before. What is happening is we are constructing this machine that's all. So, this is the diagonalization part and this is why it is called diagonalization. So, you can read up more if you are interested. So, this is the flipping of the output. So, when you feed the its own description as the input M prime will reject if it accepts its own description but then D flips M primes output.

So, it ends up doing the opposite of what it is supposed to end up doing. So, this is the contradiction here. So, this means that contradiction to what. So, everything here was a natural consequence of what we did assuming that there is a machine that decides $A_{TM} A_{TM}$. So, that is the only assumption that we made. So, consequently $A_{TM} A_{TM}$ is undecidable this assumption was wrong and hence $A_{TM} A_{TM}$ is undecidable. So, $A_{TM} A_{TM}$ is undecidable.

So, just to quickly recap what we did. D was to simulate M on its own description or rather we ran $M' M'$ with M and its own description and then we flip the output. So, when this M is replaced, this M and <M> the description of M, the part that I am circling in green, when this is replaced by D and its own description, then we go to a contradiction.

(Refer Slide Time: 14:19)



If D rejects ⟨D⟩, accepts ⟨D⟩. Contradiction. $\Rightarrow A_{TM}$ undecidable.

Theorem: Let $t_1(n), t_2(n)$ be "time constructible" functions such that $t_1(n) \log t_1(n) \in o(t_2(n))$, then

$DTIME(t_1(n)) \subsetneq DTIME(t_2(n))$.

Proof: We will show a language L such that that $L \in DTIME(t_2) \setminus DTIME(t_1)$.

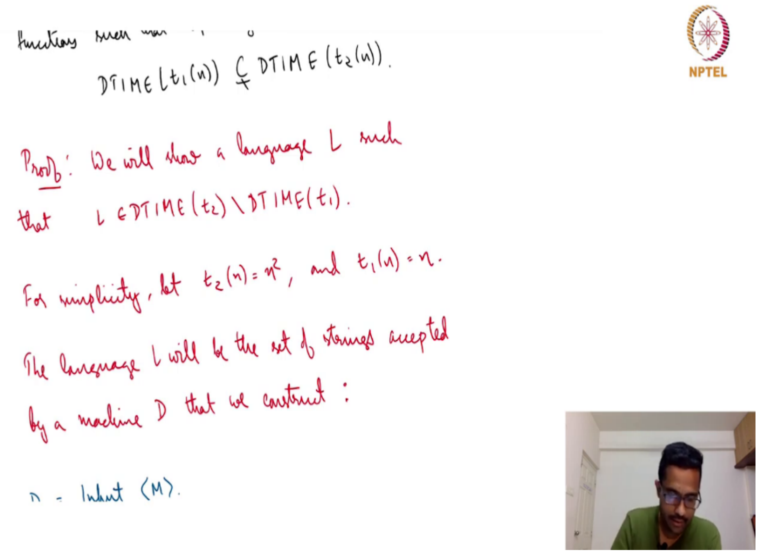
Let $t_2(n) = n^2$, and $t_1(n) = n$.

So, let us now try to see what is the time hierarchy theorem? So, time hierarchy theorem follows pretty much the same proof except that we will bring in a time factor into this whole thing. So, it will also follow diagonalization. It will also follow pretty much the same thing except that we will also start looking at the time and we will show a contradiction.

So, once again statement of time hierarchy theorem if $t_1(n)$ $t_1(n)$, $t_2(n)$ $t_2(n)$ are time constructible functions such that $t_1(n) \log t_1(n)$ $t_1(n) \log t_1(n)$ is contained is $o(t_2(n))$ $o(t_2(n))$ then DTIME($t_1(n)$ $t_1(n)$) is contained in DTIME($t_2(n)$ $t_2(n)$) which is not that difficult to see but it is strictly contained. Meaning there is some language or there are some languages that are in DTIME($t_2(n)$ $t_2(n)$) but which is not there in DTIME($t_1(n)$ $t_1(n)$) and for the proof we will demonstrate the existence of such a language.

And just for the ease of explanation we will just stick to some simple specific functions because it is just simpler to explain. Let us say $t_2(n)$ $t_2(n)$ is n^2 n^2 and $t_1(n)$ $t_1(n)$ is n which is fine because n log n is $o(n^2)$ $o(n^2)$. So, this fits the requirements of the time hierarchy theorem. So, we will show it for this n and n^2 n^2 and the proof is exactly the same it is just that for simplicity we are it is easy to explain.

(Refer Slide Time: 16:00)



functions such that

$\text{DTIME}(t_1(n)) \subsetneq \text{DTIME}(t_2(n))$.

<u>Proof</u>: We will show a language L such that $L \in \text{DTIME}(t_2) \setminus \text{DTIME}(t_1)$.

For simplicity, let $t_2(n) = n^2$, and $t_1(n) = n$.

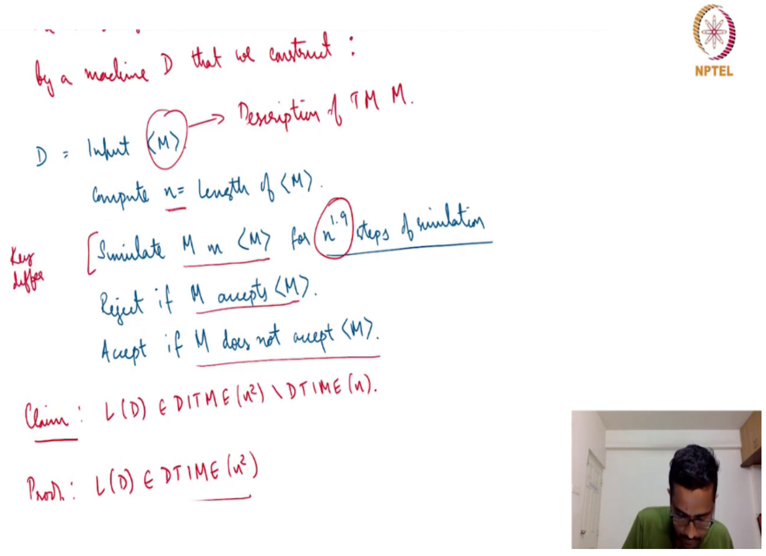
The language L will be the set of strings accepted by a machine D that we construct:

D - Input $\langle M \rangle$.

And the language L that we construct that is in $DTIME(n^2)$ $DTIME(n^2)$ but not in DTIME(n), we will not explicitly give what language it is, rather we will construct a turing machine D and whatever is recognized by that turing machine D will be the language L that is in DTIME(n^2 n^2) but not in DTIME(n).

So, what is D? So, again like I just said for the for the diagonalization of decidability this is very, very similar.

(Refer Slide Time: 16:54)



by a machine D that we construct :

D = Input $\langle M \rangle \rightarrow$ Description of TM M.

Compute n = length of $\langle M \rangle$.

Key differ [Simulate M on $\langle M \rangle$ for $n^{1.9}$ steps of simulation

Reject if M accepts $\langle M \rangle$.

Accept if M does not accept $\langle M \rangle$.

Claim : $L(D) \in DTIME(n^2) \setminus DTIME(n)$.

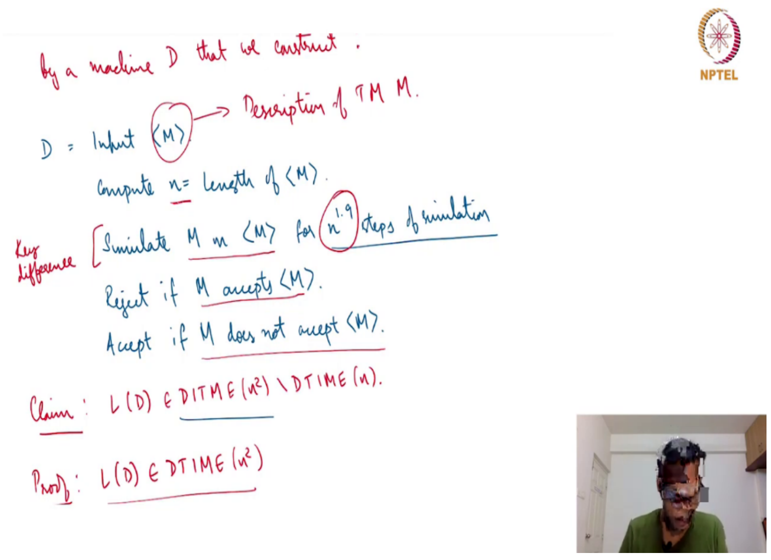
Proof : $L(D) \in DTIME(n^2)$

So, D again takes a description of a turing machine M as the input and then it computes the length of the description <M>. So, let n be the length of the description. Now what we do is we simulate again this is very similar, earlier we said that M is ran on its own description. So, now we are simulating M on its own description and the simulation we run for $n^{1.9}$ $n^{1.9}$ steps.

So,why $n^{1.9}$ $n^{1.9}$ because it is some number that is between n^2 n^2 and n. So, we simulate M on its own description for $n^{1.9}$ $n^{1.9}$ steps of the simulation. So, the total simulation will happen for $n^{1.9}$ $n^{1.9}$ steps'- time. This is the amount of time that we are willing to devote for the simulation. So, in that much time $n^{1.9}$ $n^{1.9}$ steps whatever happens, happens. We if M accepts its own description we reject. D rejects if M does not accept its own description. So, the three possibilities M can accept M its own description, the second possibility - may reject its own description and the third possibility is that we run out of time because we are only doing this for a fixed number of steps.

So, if M does not accept. So, if it rejects or does not accept its own description then D accepts the description of M. So, again there is a flipping happening if M accepts <M>, D rejects and if M does not accept <M>, D accepts. So, again the only thing here is that earlier we had a decider and then we flip the output. Here we are doing a simulation. So, this is the key part that is different. This the step is the key difference here. We have a simulation that is a time bounded simulation and then we flip the output.

(Refer Slide Time: 19:15)



by a machine D that we construct.

D = Input $\langle M \rangle \rightarrow$ Description of TM M.

Compute n = length of $\langle M \rangle$.

Key difference [Simulate M on $\langle M \rangle$ for $n^{1.9}$ steps of simulation

Reject if M accepts $\langle M \rangle$.

Accept if M does not accept $\langle M \rangle$.

Claim: $L(D) \in \text{DITME}(n^2) \setminus \text{DTIME}(n)$.

Proof: $L(D) \in \text{DTIME}(n^2)$

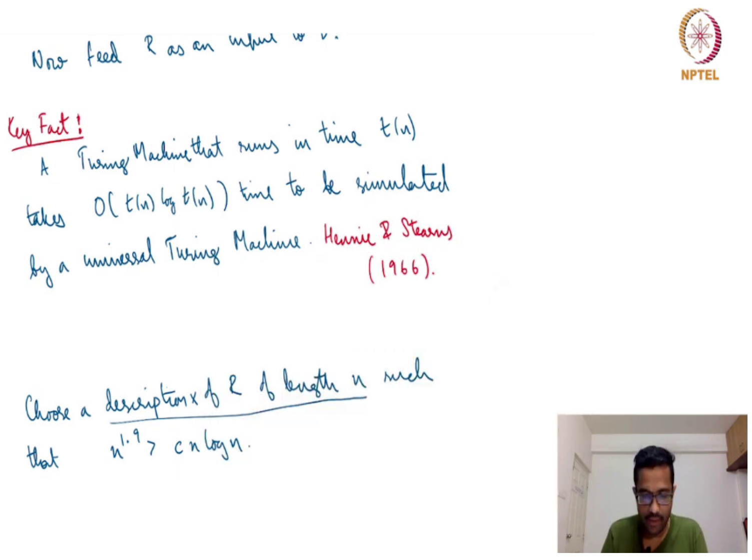
And now we claim that the language recognized by this machine the Turing machine D is in order n^2 n^2 time is recognizable in DTIME(n^2 n^2) but not in DTIME(n). First part is easy why is it in DTIME(n^2 n^2) this is easy because we run the simulation only up to $n^{1.9}$ $n^{1.9}$ steps. So, it is less than o(n^2 n^2). So, as simple as that this is easy. Second part is the harder part why is it not contained in DTIME(n)?

So, again just like earlier we proved the diagonalization we will prove. We will use diagonalization to show that why this language recognized by this Turing machine cannot be decided in time n. Let us see why? Suppose it was decidable in time DTIME(n). Now suppose we know the Turing machine D runs $n^{1.9}$ $n^{1.9}$ time and now we are saying that suppose it can be decided in order n time ie., DTIME(n).

So, maybe there is another tuning machine that decides it in order n time. Let that turing machine be called R. So, this is R, R is a decider that decides the language decided by D in order n time.

Now let us feed R as an input to the machine D. So, now instead of earlier we fed D its own description. we will feed D the description of R and then we will see what will happen.

(Refer Slide Time: 21:34)



Now feed R as an input to D.

Key Fact!
A Turing Machine that runs in time $t(n)$ takes $O(t(n) \log t(n))$ time to be simulated by a universal Turing Machine. Hennie & Stearns (1966).

Choose a $\underline{\text{description } x \text{ of } R \text{ of length } n}$ such that $n^{1.9} > c n \log n$.

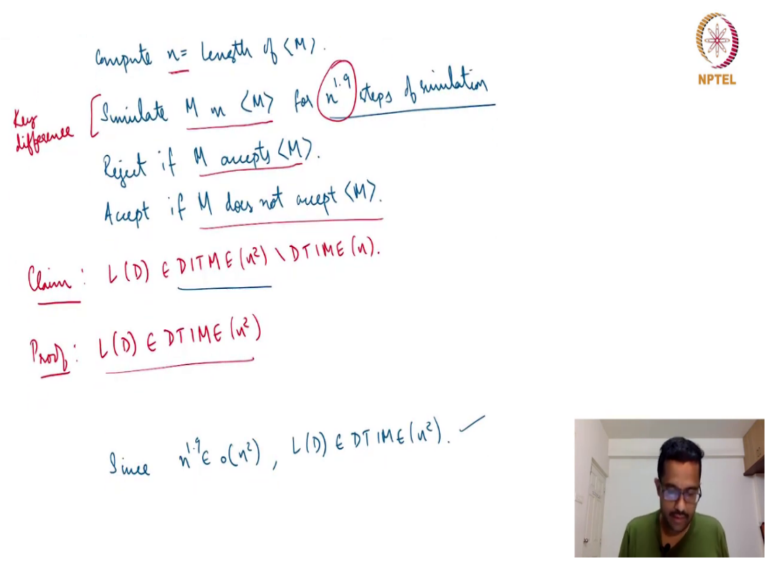
So, now we will feed R as an input to D. So, the key thing here is if you may have noticed that the statement of time hierarchy theorem there is a log thing happening here. Only when $t_1(n)$ $t_1(n)$ is contained in $t_1(n) \log t_1(n)$ $t_1(n) \log t_1(n)$ which is $o(t_2(n))$ $o(t_2(n))$, can we say this. Meaning if $t_1(n)$ $t_1(n)$ is n and $t_2(n)$ $t_2(n)$ is n log n or let us say $n \sqrt(\log n)$ $n \sqrt(\log n)$ then we cannot apply time hierarchy theorem because $t_1(n) \log t_1(n)$ $t_1(n) \log t_1(n)$ t will be not $o(t_2(n))$ $o(t_2(n))$.

So, this case this way it is not possible. So, what I am saying is that we have this log requirement here. Why does the log requirement appear because how does the simulation happen. So, how does one machine simulate another turing machine. So, earlier we use in theory of computation course you might have seen that we used to just say we simulate this machine on that machine and so on.

But here we are bothered about the time. So, we need to keep track of how much time it takes. So, a turing machine that runs in time t(n) actually needs t(n)log t(n) time to be simulated. So, it requires t(n)log t(n) time to be simulated by a universal turing machine. So, universal turing machine is something that that can simulate everything. So, this is a result by Hennie and Stearns from 1966. And basically a machine that runs in t(n) time for another machine to simulate the same thing it takes t(n)log t(n) time.

So, for simulating one machine by another the time increases logarithmically by a logarithmic factor. So, this is the reason why we have a logarithmic factor in the statement of the time hierarchy theorem as well. If this was not there then we would also donot need to have the log factor in the time hierarchy theorem. This log goes all the way to that log. So, and this is the best that we know Hennie & Stearns results has not been improved from 1966. So, now how does it follow?

(Refer Slide Time: 24:06)



Compute $\underline{n} =$ length of $\langle M \rangle$.

Key difference [Simulate $\underline{M \text{ on } \langle M \rangle}$ for $\underline{\boxed{n^{1.9}} \text{ steps of simulation}}$

Reject if $\underline{M \text{ accepts } \langle M \rangle}$.

Accept if $\underline{M \text{ does not accept } \langle M \rangle}$.

$\underline{\text{Claim}}$: $L(D) \in \underline{DITME(n^2)} \setminus DTIME(n)$.

$\underline{\text{Proof}}$: $\underline{L(D) \in DTIME(n^2)}$

Since $n^{1.9} \in o(n^2)$, $L(D) \in DTIME(n^2)$. ✓

So, now again the goal here is what we have now is a turing machine D which runs in o(n^2) time or endpoint n^1.9 time we assume that it has an order n decider and that decider is called R and now we are going to feed the description of R to D. Amongst all the descriptions we pick a description x. So, let n be the length of the description of x.

So, we said that. So, obviously n power 1.9 is bigger than n log n but we will choose n such that um. So, maybe for small values of n may be the right hand side is bigger. So, we will choose n such that in left hand side, the asymptotic's kickin. So, maybe the at t the initial part smaller values of n, it could be that the right hand side is bigger. So, we will choose beyond a point the left hand side will be bigger. So, we will choose n like that.

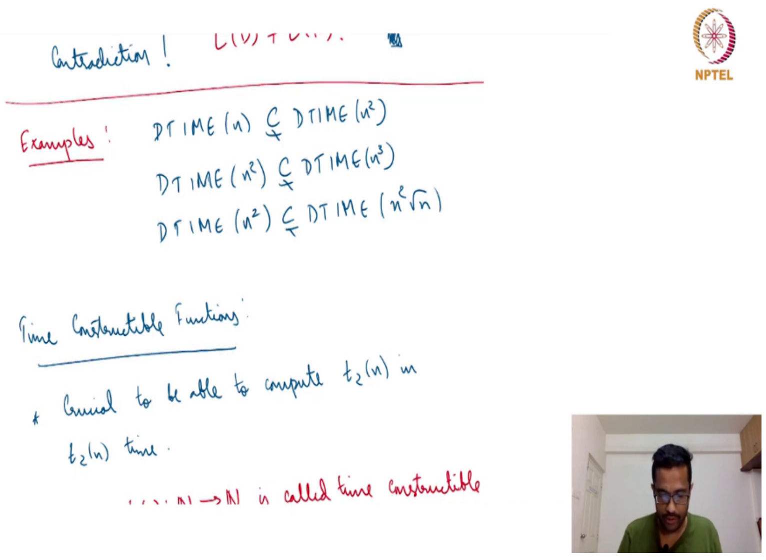
So, now we run D on the description of R or a specific description of R called x. Now what happens. So, you are giving x as the input of D. So, what is D does is it simulates R; recall that x is a description of R. So, D will simulate R on x for $n^{1.9}$ $n^{1.9}$ steps. And R is assumed to be linear time. So, by O(n) time R would have decided either way and even in taking into account the logarithmic overhead required for the simulation n log n is smaller than and even c n log n is smaller than $n^{1.9}$ $n^{1.9}$.

So, we will certainly have known whether R accepts or R rejects are not this being able to decide and terminating is will not happen because we know that R terminates in order n time and we have accounted for enough time. So, now D rejects x if r accepts and D accepts x if R rejects. So, now what does it mean D and R were supposed to recognize the same language. So, that is how we picked R.

R was picked to be another machine which decides the same language as D in a smaller time. But now we are we are seeing in x that does not behave the same way in D and R. So, whatever D does with whatever R does with x, D does the opposite. So, this is a contradiction L(D) is not equal to L(R). Again this is a diagonalization proof and diagonalization proofs can be tricky.

So, let me just go over it once more so D is the following machine given an input which is a description of a turing machine it first computes the length of that input. So, given M it computes the length of M and simulate the machine on its own description. M is simulated on its own description for $n^{1.9}$ $n^{1.9}$ steps and D rejects if M accepts and D accepts if M does not accept, so, basically flipping the output of M.

(Refer Slide Time: 28:13)



Contradiction !

Examples : $DTIME(n) \subsetneq DTIME(n^2)$

$DTIME(n^2) \subsetneq DTIME(n^3)$

$DTIME(n^2) \subsetneq DTIME(n^2\sqrt{n})$

Time Constructible Functions :

* Crucial to be able to compute $t_2(n)$ in $t_2(n)$ time .

$\mathbb{N} \to \mathbb{N}$ is called time constructible

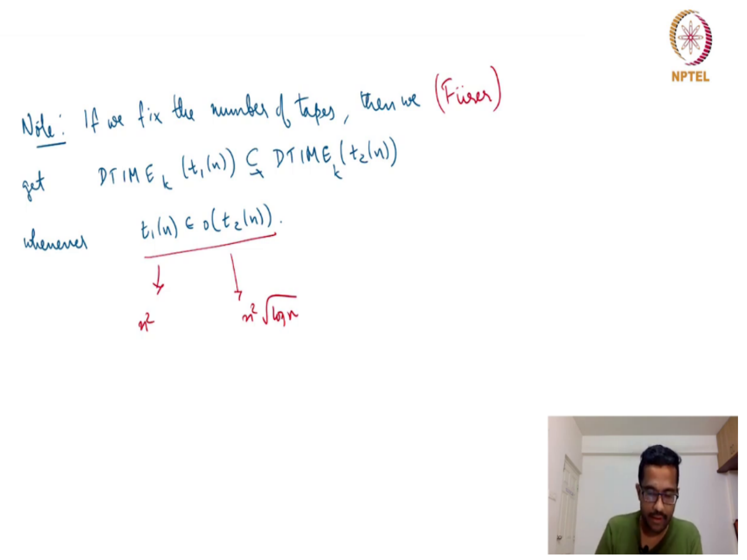
And this the language decided by D we claim that it is in order it is in $DTIME(n^2)$ $DTIME(n^2)$ but not in DTIME(n) why? So, it is in $DTIME(n^2)$ $DTIME(n^2)$ because the simulation runs in $n^{1.9}$ $n^{1.9}$ time and it is not in DTIME(n) because we assume the quantity assume the opposite and arrive at a contradiction. So, we assume that it is in DTIME(n) through a machine R. So, R is a decider for D which runs in DTIME(n).

And now we feed a specific input specific description of R called x to D and this specific description is chosen such that the length of the description is n and we can complete the simulation in $n^{1.9}$ $n^{1.9}$ time. So, now what does D do on when it takes x as input , recall that x is a description of R. So, it will run R on x and then do the opposite and we know that R will terminate the computation on x and D will do the opposite.

So, by definition D does the opposite of what R does on the same input but then R was chosen to be or R was decided to be something that recognize or decides the same language as D and that is a contradiction. Just some just some so, this is the time hierarchy theorem. So, why what is hierarchical about time hierarchy theorem. So, with this I can show that D time we have already seen the DTIME(n) is contained in but not it is it is not equal to meaning the in the contained is strict in DTIMES(n^2 n^2) is contained in the time n^3 n^3 anything.

Maybe I can say even more things DTIME(n^2 n^2) is even contained in the DTIME($n^2\sqrt{n}$ $n^2\sqrt{n}$) because again you can work out with it. So, like that we can. So, if there is significant gap between $t_1(n)$ $t_1(n)$ and $t_2(n)$ $t_2(n)$ then there is a language that that can be decided in $t_2(n)$ $t_2(n)$ but not in $t_1(n)$ $t_1(n)$ and these languages are usually. So, you can see how non-explicit this is we are not specifically telling which language it is but we are saying that this is a language. Now finally I will just conclude by saying what are time constructable functions?

(Refer Slide Time: 31:03)



Note: If we fix the number of tapes, then we (Fürer)

get $\mathrm{DTIME}_k(t_1(n)) \subsetneq \mathrm{DTIME}_k(t_2(n))$

whenever $\underbrace{t_1(n) \in o(t_2(n))}$.

n^2 $\quad$ $n^2\sqrt{\log n}$

So, the key point here is we should be able to compute $t_2(n)$ $t_2(n)$ in $t_2(n)$ $t_2(n)$ time. So, here D runs simulates M on M on $n^{1.9}$ $n^{1.9}$ steps, so, this $n^{1.9}$ $n^{1.9}$ it needs to compute. So, if $n^{1.9}$ $n^{1.9}$ itself takes n^3 n^3 time to compute then this what we said here is not correct $n^{1.9}$ $n^{1.9}$ the value may be smaller but to compute that takes n^3 n^3 time then it is not in DTIMES(n^2 n^2).

So, we need to make sure that $n^{1.9}$ $n^{1.9}$ can be computed in $n^{1.9}$ $n^{1.9}$ time and this is exactly what time constructability is. The function $t_2(n)$ $t_2(n)$ should be computable in $t_2(n)$ $t_2(n)$ time and this is what is time constructability. So, more formally a function t is called time constructable if we give n ones(1^n 1^n), a string of length n as an input to a turing machine, it takes that as input and computes $t(n)$ $t(n)$ and outputs the binary t(n) in O(t(n)) steps. So, example almost everything n, n^2 n^2 . So, n can be computed in n time. n^2 n^2 can be computed in n^2 n^2 time , and similarly 2^n 2^n ,log n etc

So, you can maybe you can try this why is this time constructable you can think about this these are all time constructable functions in fact most of whatever we can think of are time constructible functions. To demonstrate something that is not time constructable we again need to resolve diagonalization some make highly artificial functions that is about time constructability. And finally, one more thing I said that DTIME($t_1(n)$ $t_1(n)$) is strictly contained in DTIME($t_2(n)$ $t_2(n)$) if $t_1(n) \log t_1(n)$ $t_1(n) \log t_1(n)$ is contained is o($t_2(n)$ $t_2(n)$).

And I said that there is no better simulation known but there is a better simulation known due to Furer, if we fix the number of tapes. So, if we fix the number of takes to k then there is a better simulation known where and because of that we can get a strict hierarchy DTIME($t_1(n)$ $t_1(n)$) is contained DTIME($t_2(n)$ $t_2(n)$) strictly ~~content~~ contained but not equal to whenever $t_1(n)$ $t_1(n)$ is o($t_2(n$ $t_2(n$)). So, even in this case like the one that I wrote here I have not have here is that probably even when $t_1(n)$ $t_1(n)$ is even when $t_1(n)$ $t_1(n)$ is n^2 n^2 and $t_2(n)$ $t_2(n)$ is $n^2\sqrt{(\log n)}$ $n^2\sqrt{(\log n)}$ even then this containment will work.

But this is only when we when the number of tapes are fixed for both left hand side as well as right hand side and not in general. So, that is the time hierarchy theorem. So, now given any two functions like n^2, n^3 n^2, n^3 which are sufficiently separated then there is a language that can be computed in n^3 n^3 time but not in n^2 n^2 time. That is the summary of this lecture. Again diagonalization can be bit confusing just keep re-watching it and thinking about it I think after a while it will become clear, thank you.