

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

**Lecture -39**  
**Turing Machines that Take Advice**

(Refer Slide Time: 00:16)

The image shows a presentation slide titled "Lecture 39 - Turing Machines that take advice". The slide contains handwritten notes in blue and red ink. The notes are as follows:

- P/poly: Why this notation?**
- Non-uniformity helps it to decide even undecidable languages!
- Is there a way to control its power?
- Uniform circuit families:  $\{C_i\}_{i=0,1,2,\dots}$  is a
- uniform circuit family of the function
- $\cdot$  is associated in  $\cdot$

In the bottom right corner of the slide, there is a small video inset showing a man in a blue shirt speaking. The NIPTEL logo is visible in the top right corner of the slide.

Hello and welcome to lecture 39 of the course, computational complexity. So, in this lecture we will see Turing machines that take advice what are they and how they are related to some of the circuit models that we have seen so far. So, hopefully this will help address some questions that we have seen throughout this the circuit complexity part. So, one question that you may have some of you may have been wondering is like we have always been calling this class P by poly.

What is the reason for this notation P slash poly? And another thing that we have noted is that non uniformity is such a big, it has so much power it is a non-uniformity meaning the capability to have different circuits for different input sizes that can do a lot of things. So, that can even allow P by poly to decide undecidable languages. So, non-uniformity is like a it is like a big game changer.

So, now is there a way to kind of restrain the power of non-uniformity. Is there a way to control its power? This is another question that some of you may have wondered. Is it is non-uniform circuits is that the correct model or is there a better model for circuits?

**(Refer Slide Time: 01:43)**

undecidable languages;  
 \* Is there a way to control its power?

Uniform circuit families:  $\{C_i\}_{i=1,2,\dots}$  is a  
 \* uniform circuit family of the function  
 $f^i \rightarrow C_i$  can be computed in \*  
 \* - could be P, LOGSPACE,  
 or any other class.

Example: P = P-uniform circuit family



So, there happens to be a better model called uniform circuit families. So, a set of circuits is called a uniform circuit family if there is a function that can be computed by a certain computational power. So, that functions describing the circuits. So, that function should take us input a unary representation of the input length. So, for length 10 to obtain length 10 circuits the function should take as input 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, an input of 10 ones.

And output the description of the circuit family or output the description of the circuit that will take 10 long inputs. So, this is the requirement and this function should be computed in a certain computational power. So, it if you say P uniform circuit family this function should be computed in polynomial time if you say log space uniform circuit family. So, this function should be computed in log space.

So, this could be log space P or whatever it could be some other classes too. So, this is what is meant by uniform circuit family. So, I am just writing star uniform and where star can be replaced with any bounded computational power class like P or log space or something else and

it should take as input a union representation of the input length and output the description of the circuit.

So, any circuit can be described by saying how many gates are there? What are it? What are the gate types? Where do the inputs come from and so on. So, basically implicitly describing the graph.

(Refer Slide Time: 03:48)

The slide contains handwritten notes in red and green ink. At the top, it says "x - could be ... of any other class." with a line underneath. To the right is the NPTEL logo. Below that, it says "Example: P = P-uniform circuit family" and "P = logspace-uniform circuit family". A circled "P/poly" is written next to "P-uniform circuit family" with a checkmark and the word "submitted" next to it. Below this, it says "Exercise: Try to show the above two." At the bottom, there is a definition: "Turing Machines that take advice" followed by "DTIME(t(n))/A(n): The class of languages accepted by a DTIME(t(n)) machine when given".

So, this is what is called uniform circuit families. So, now uniform circuit families means that there should be a way to generate the circuits of each input length in a using some kind of an algorithm or some kind of a computational power, some kind of an algorithm and this will kind of reduce the power of the circuits meaning it will remove the non uniformity. Because again the by the name itself indicates that these are called uniform. So, this does not have the non-uniformity because you have a way to generate given any input length.

You have a way to generate a circuit family in an algorithmic manner in this in a bounded in a controlled power using a controlled algorithmic power. And then now it is just about the circuit evaluating a certain input. So, for instance if you have a P uniform circuit family. So, for first think of P uniform P by poly you have polynomial sized circuits that are generated by a P uniform function meaning if there is a function that takes as input the length of the input of the circuits and generate the circuit description.

And suppose the circuit description is a polynomial in size so it will be a  $P$  uniform  $P$  by poly and this happens to be equal to  $P$ . Why is that? We have already seen that  $P$  is contained in  $P$  by poly. So, that still holds even if it is  $P$  uniform  $P$  by poly and now to show the other way containment you could use a polynomial time machine to generate the input description and also to evaluate the circuits.

So,  $P$  is equal to  $P$  uniform  $P$  by poly. In fact we can skip the mentioning, this mention can be omitted because any circuit family that can be described or generated in polynomial time that will necessarily be of polynomial size because in polynomial time we cannot hope to output an exponential size circuit. So, if you just say  $P$  c,  $P$  uniform circuit family that itself has to be equal in power to or like the class of  $P$  uniform circuit families must be equivalent in power to  $P$ .

Similarly you can also verify that log space uniform circuit family even that is equivalent in power to  $P$ , the class of languages that are decided by log space uniform circuit families is equivalent to the class of polynomial and time algorithms. If you again look at the proof of the Cook-Levin theorem, the circuit family that we obtain is actually a log space uniform circuit family. So, that is another thing to note.

So,  $P$  is actually equal to  $P$  uniform circuit family as well as log space uniform per circuit family. So, we may mention  $P$  log space uniform  $P$  by poly but that is not really required. Because again if something is generated in logarithmic space then it must necessarily have polynomial size, it cannot have more than polynomial size. So, you can try to work out the about two I am not getting into the details but it is rather straight forward. Now let us just define Turing machines that take advice.

**(Refer Slide Time: 07:29)**



example:  $P = \text{logspace-uniform circuit family}$

Exercise: Try to show the above two.

Turing Machines that take advice

$\text{DTIME}(t(n))/a(n)$ : The class of languages accepted by a  $\text{DTIME}(t(n))$  machine when given an advice  $a(n)$ .

$L \in \text{DTIME}(t(n))/a(n)$  if  $\exists$  TM  $M \in \text{DTIME}(t(n))$  and  $\exists$  advice  $a(n)$  such that  $|a(n)| \leq a(n)$ , and



So, what are Turing machines that take advice? So we will denote them as  $\text{DTIME } t(n) \text{ slash } a(n)$  and this is the genesis of the notation  $P$  by poly. So, the class of language is so these are  $\text{DTIME } t(n) \text{ slash } a(n)$  these are Turing machines that take advice. What are they? These are the  $\text{DTIME } t(n) \text{ slash } a(n)$  is the class of languages that is accepted by a  $\text{DTIME } t(n)$  Turing machine which has an advice string available to it, the advice string is  $a(n)$ .

**(Refer Slide Time: 08:16)**



$\text{DTIME}(t(n))/a(n)$ : The class of languages accepted by a  $\text{DTIME}(t(n))$  machine when given an advice  $a(n)$ .

$L \in \text{DTIME}(t(n))/a(n)$  if  $\exists$  TM  $M \in \text{DTIME}(t(n))$  and  $\exists$  advice  $a(n)$  such that  $|a(n)| \leq a(n)$ , and  $x \in L \iff M(x, a(n)) = 1$ . fixed for each input length  $n$

Example:  $\text{UNHALT} \in P/1$

$P/1 = \bigcup_n \text{DTIME}(n^k)/1$



So, to be more formal  $L$  is in  $\text{DTIME } t(n) \text{ slash } a(n)$  if there is a deterministic Turing machine  $M$  that runs in  $\text{DTIME } t(n)$  or that runs in time  $t(n)$  and it gets its input in addition to the actual input it gets an advice string  $a(n)$  where  $a(n)$  is of length at most  $a(n)$ . So, this is a type of it should be a  $n$

and where  $n$  is some function, it could be polynomial, exponential, logarithmic whatever. So, the  $\alpha_n$  is the advice string and that the length of the advice string should not be more than a  $n$ .

So, this is the definition of a tutoring machine that takes a device. So, this may seem similar to NP, definition of NP where it had access to a verifier model of NP where we had a proof string or witness string or certificate string for each input. The main difference here is that so there the two differences, there the length of the advice string or sorry not otherwise the witness or certificate string was polynomial bounded by a polynomial in the input of the length of the input.

Here the length of the advice string is bounded by some function  $\alpha_n$ , this could be anything it could be logarithmic, it could be polynomial, it could be exponential. Some function  $\alpha_n$ , you could define it just changes the parameter in the definition. Second thing rather this is the more important definite point is that there the certificate could change with the input. For each input we could have a different certificate, for each graph I could have a different three coloring, for each Boolean formula I could have a different satisfying assignment.

Whereas here there is still some variation with input but this the advice string  $\alpha_n$  is fixed. So, use a different colour, this is fixed for each input length. So, once an input of length  $n$  is given the  $\alpha_n$  is fixed. There we did not have that where we could have a separate witness or certificate string for each input even two different inputs of the same length could have different certificate strings.

Whereas here the advice string is just depend; on the length of the input it cannot keep changing based on the input. So, for a fixed length whatever may be the input the advice string has to be the same, it can only be dependent on the input length. So, the  $\alpha_n$  is fixed for a specific  $n$ . So, these are the main differences, the second difference is rather more important. So, we cannot it does not have as much power or it does not have the kind of power that NP has.

Because it is fixed for a certain input size and  $x$  is in the language if the Turing machine accepts it along with the help of the advice string. So, the Turing machine should accept  $x$  and along with the  $\alpha_n$  of the corresponding input length. So, just to give some idea we saw  $u$  halt which is the

union version of the halting problem, this actually was in P slash 1. The advice thing here was just a one bit of advice thing. So, for each input length we need to know whether that input length k is in the language or not. So, the idea string could be one bit.

We could say zero when the input length is input like that there is no string of length k in the language and you could say 1 when there is only one string that is one power k that is in the language. So, U halt is actually in P slash 1, in fact it is actually it is in P slash 1. But you could not if there is a class of smaller size than P then we could have that as well.

**(Refer Slide Time: 12:55)**

Handwritten notes on a whiteboard:

- Top right: NPTEL logo
- Top left:  $x \in L$
- Top center: Example: UHALT  $\in P/1$
- Top right: input length  $n$
- Center: Theorem:  $P/poly = \bigcup_{c,d} DTIME(n^c)/n^d$
- Bottom left: Proof: If  $L \in P/poly$ , then the DTM simulator evaluates the circuit given its description.
- Bottom center: If  $L \in DTIME(n^c)/n^d$ , then there is a DTM that takes  $x$  and  $x^{1/d}$  as input, where  $x^{1/d}$  is a  $d$ -bit string.
- Bottom right:  $DTIME(n^c)/n^d$  ↑ describe the circuit for length  $n$
- Bottom right: Small video inset of a person in a blue shirt.

And P by poly, we have already seen is actually equal to d time n power c divided by n power d or d time n power c n power t. So, now you could see why the notation P by poly has used. Because the numerator or the sort of numerator that we have is d times n power c which is if you take the union of over all the c it is just the class P, d time n power c. And the denominator is n power d which is again a polynomial so that is why we call it P by poly. This is the genesis of the word P by pol.

So, let us see why this is the case. So, the claim is that P by poly is the class of all languages that are decided by polynomial size circuits. And now I am saying that this is the same as the class of all languages decided by Turing machines that take a certain amount a polynomial length advice.

Let us see why that is the case. Suppose  $L$  is in  $P$  by poly which means there is a polynomial size circuit family that decides  $L$ .

Now why is it in  $d$  time  $n$  power  $c$  slash  $n$  power  $d$  which means that given an input that the Turing machine has to decide. So, what will the Turing machine do? The Turing machine could notice that the tuning machine has to be same for the entire language. But it could use an advice string or the advice string could vary based on the input length. So, the very obvious or kind of immediate thing to guess will be the description of the tuning description of the circuit.

So, for each input length there is a different circuit. So this  $n$  power  $d$  could describe the circuit for length  $n$ . So, for a certain length input in power  $d$ , the polynomial size input string could describe the circuit for length  $n$  and then the deterministic Turing machine merely simulates the input evaluates the circuit on that input. So, simulates or maybe evaluates is a better word, maybe evaluates the circuit at the input. So, this is why  $P$  by poly is contained in  $DTIME$   $n$  power  $c$  slash  $n$  power  $d$ .


**(Refer Slide Time: 15:55)**

*Proof: If  $L \in P$  by poly, then the NIM <sup>number</sup> evaluates the circuit given its description.*

*describe the circuit for length  $n$*

*If  $L \in DTIME(n^c)/n^d$ , then there is a DTM that takes  $x$  and  $x_{adv}$  as input, where  $|x_{adv}|/|x| \leq n^d$ . Using ideas in Cook-Levin theorem, we can build a circuit for all  $x$  of an input length  $n$ , hard-wiring  $x_{adv}$ .*

*proof of Cook-Levin*




Now let us see the other direction. Suppose there is a Turing machine that takes a device or suppose  $L$  is in a Turing machine that takes a device. Now  $L$  is in  $DTIME$   $n$  power  $c$  slash  $n$  power  $d$ . So, this means there is a Turing machine that takes  $x$  as input and for each input length it has a different advice string, where the length of the advice string is bounded by a polynomial.



So, length of  $\alpha x$  which is the advice on the input  $x$  is upper bounded by  $n^d$ . So, it is  $\alpha x$  is the length of the input.

So, it should actually say the length of  $\alpha x$  which is  $\alpha$  length of  $\alpha n$  is upper bound by  $n^d$ . Now we could recall again we could once again recall the proof of Cook Levin theorem. So, there is a Turing machine that runs in polynomial time  $d$  time  $n^c$  that takes input  $x$  and they takes a certain  $\alpha n$  and we could build a circuit that evaluates that corresponds the decision of process of the steering machine.

So, that circuit will take  $x$  and  $\alpha x$  or  $\alpha n$  as inputs, but then  $\alpha n$  is fixed for a certain input length. So, we could we could actually hard code the description of  $\alpha n$  or hard code that  $\alpha n$  into that circuit that may possibly simplify the circuit. So, we can build a circuit for this as in by the proof of Cook Levin theorem. So, we are really milking the proof of Cook Levin theorem again and again by noticing that there is computation is local so that you could replace it with a constant size circuit and so overall it is a polynomial size circuit.

So, you could build a circuit and then you hard code  $\alpha$  into that circuit and then that is it you have the circuit that decides whether if  $x$  is in the language or not and obviously that circuit will be polynomial size. So, that is why  $d$  time  $n^c$  slash  $n^d$  is contained in P by poly and that is all that I have to say in this lecture. So, we saw what are uniform circuit families, it could be any uniform, P uniform, log space uniform and then we saw the definition of Turing machines that take advice these are Turing machines that run in time  $t n$ .

But has access to an advice string of length  $a n$  of length at most  $a n$ . And this advice string only is a function of the input length not the input itself. And it could vary with the input length but will be same once the input length is fixed. And then we saw that P by poly is in fact  $d$  times  $n^c$  slash  $n^d$ , union of over all the  $c n$  and that is all I have for this lecture.