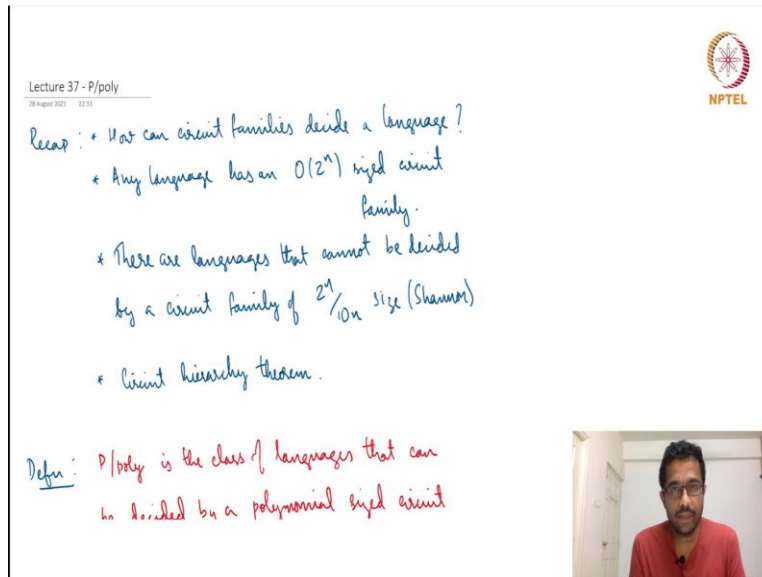


**Computational Complexity**  
**Prof. Subrahmanyam Kalyanasundaram**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**

**Lecture - 37**  
**Complexity Class: P/Poly**

**(Refer Slide Time: 00:15)**





Lecture 37 - P/poly  
28 August 2021 12:51

**Recap:**

- \* How can circuit families decide a language?
- \* Any language has an  $O(2^n)$  sized circuit family.
- \* There are languages that cannot be decided by a circuit family of  $2^n/10n$  size (Shannon)
- \* Circuit hierarchy theorem.

**Defn:** P/poly is the class of languages that can be decided by a polynomial sized circuit

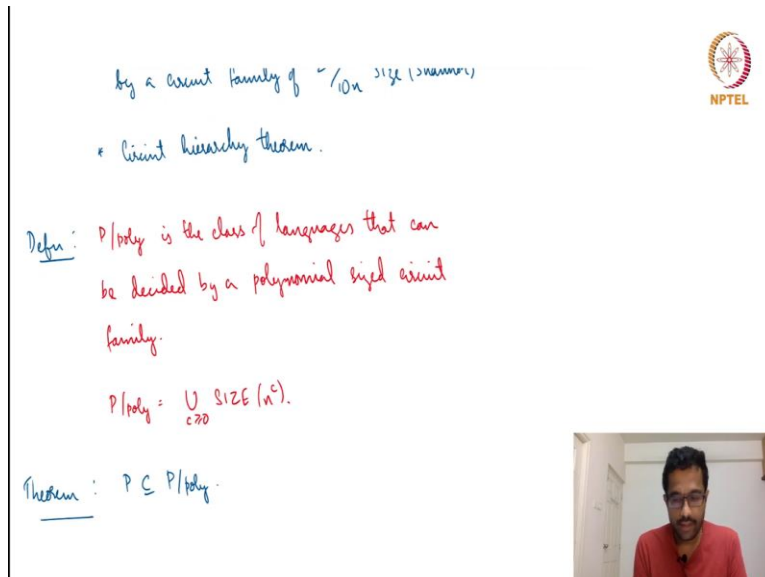


Hello and welcome to lecture 37 of computational complexity. In this lecture we will see the circuit complexity class P by poly. So, just to recap what we have seen in circuit so far, we have seen how circuit families can decide a language even though it is a model that computes a function how we can use it to decide a language. So, this was done by having a circuit family one for each input length and by associating the inputs for which it gives one as an output as those in the language.

We saw that any language has an order  $2^n$  sized circuit family. In fact I mentioned that this could be improved to  $2^n/n$  that was Lyapunov theorem. And we also saw Shannon's theorem which said that there are languages that cannot be decided by a circuit family of size  $2^n/10n$ . And in fact, what was even more surprising is that almost all the languages, almost all the functions are hard meaning they cannot be computed in a circuit family of this size.

And we also saw the circuit hierarchy theorem which said that if you have a circuit of bigger size, then it can potentially compute more functions than what could be computed by a circuit of a smaller size.

**(Refer Slide Time: 01:56)**





by a circuit family of  $\sim 10^n$  size (numbers)

\* circuit hierarchy theorem.

Defn: P/poly is the class of languages that can be decided by a polynomial sized circuit family.

$$P/poly = \bigcup_{c \geq 0} SIZE(n^c).$$

Theorem:  $P \subseteq P/poly$ .



So, today in this lecture we will define a class called P by poly which is simply the class of languages that can be decided by polynomial size circuits. So, polynomial size circuits mean maybe for simplicity we will just use AND, OR and NOT gates and it is immaterial whether we allow like fan in  $n$  or constant fan in. Because a fan in a constant fan in a bigger fan and can be replaced by a tree of constant fan and gates, and it will still be polynomial sized.

So, in other words P by poly can be written as union size  $n$  power  $c$  for  $c$  equal to 1, 2, 3 and so on. So, it is one may think that it is a circuit equivalent of the class P which was the class of all the languages that can be decided by deterministic polynomial time Turing machines. However, so let us see if that is indeed the case or let us see if how these two classes compare P and P by poly.

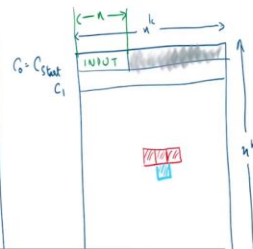
**(Refer Slide Time: 03:12)**

Theorem:  $P \subseteq P/poly$ .

Proof: Let  $L \in P$ . Then it is decided by a DTM  $M$  in  $n^k$  time, for a constant  $k$ .

We could write down the computation of  $M$  as a computation table.

like in the proof of



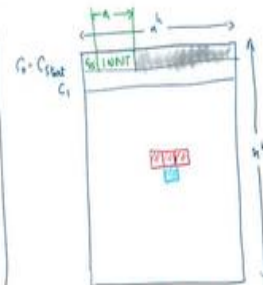
So, first we will see that  $P$  is contained in  $P$  by poly. So, as always, we will take an arbitrary language in  $P$  and show that it is contained in  $P$  by poly which is the standard way of showing containment or inclusion of sets. So, let  $L$  be a language in  $P$ , so again just to state this is the Cook-Levin theorem there will be a language in  $P$  that means there is a deterministic Turing machine  $M$  that computes it in decides it in  $n$  power  $k$  time,  $n$  power  $k$  time for some constant  $k$ .

**(Refer Slide Time: 03:50)**

$n^k$

We could write down the computation of  $M$  as a computation table.

like in the proof of Cook-Levin theorem, we note that each cell is only directly dependent on the three cells above it.



Computation table



If you recall the proof of Cook-Levin theorem we could draw a computational table or computational tableau. So, this is what we have here computation table or tableau where we start with the first starting configuration where actually most of it is the input and may be the starting

state. So, maybe I should include the starting state I have just written input. So, the starting state  $q$  start followed by input and maybe I should just redraw this, so this part will be  $n$  for the input.

So, this is the starting configuration then followed by the next configuration next configuration and so on. This is what we did in the Cook-Levin theorem, the proof of the Cook-Livingston theorem. And the proof crucially hinged on the fact that the computation in a Turing machine is an extremely local phenomenon. So, the head is somewhere in the looking at the tape and the head moves left right and just one step on left or one step right or stays.

In any case it cannot make changes very far away it can only make changes in the next cell to the left or to the right when you look at it in the configuration sense only the if you look at the head location only the place where the head is or the immediately left location or the immediately right location. These are the places where the configuration can possibly change and we also had this notion of 2 by 3 windows where we tried to see where some changes happen.

So, again, we will in this theorem also we will make use of the fact that the computation in a Turing machine is a local phenomenon. So, what we have to do here? We have a Turing machine that decides a language  $L$  in polynomial time, so we can draw the computation table. Now we want a circuit, so we want to again we want to show that  $P$  is contained in  $P$  by poly. So, we want to show a polynomial size circuit family for  $P$  by poly.

So, all that we will focus on is to obtain, so I said that the computation in the Turing machine is a local phenomenon, all I will focus on is to get a circuit to simulate or to make this local computation. So, this as I have indicated here the blue cell here which is in a certain row is only going to be affected by the red cell directly above it, directly above and left and directly above and right. These three red cells will completely determine the blue cell.

So, it is just a function of these three things and so on like and this is the case for all the cells. So, the blue cell is just some cell that I picked out in the computation table. So, now I could have a Boolean formula that given the contents of the red cells, I could have a formula or if I break

down the content of the blue cells into binary symbols or bits. It may be long it may not be one symbol but it could be a big string.


But I could have a Boolean circuit or circuits that generate this; the content of this blue cell. And as we have seen in the case of cook levin theorem proof this computation will be a constant size computation meaning this circuit will be a constant size circuit. So, in fact it is even more simpler than cook levin and theorem proof because in the case of cook levin theorem, we had we were trying to deal with a non-deterministic tutoring machine.

Whereas here we; are just dealing with a deterministic Turing machine. So, we do not have to like once the red cells are given the blue cell will be determined completely by the transition function. There is no possibility of multiple paths and all that and this circle part that is the red cells and the blue cell below it we use it we can get a constant sized circuit to determine that. And now the computation table if you see the width of the computation table is  $n^k$  where  $k$  is running time.

That is because the Turing machine can traverse up to  $n^k$  cells in the tape. Because that is the time that it has it cannot traverse longer more than that because this moves one step per time. And we know that the computation number of steps is  $n^k$  because that is the running time bound. So, the height and the width are both  $n^k$ , so there are polynomials  $n^{2k}$  many cells and the content of each cell can be determined by a constant size circuit.

So, there are  $n^{2k}$  cells whose content have to be determined may be except for the first row. And each of these  $n^{2k}$  cells is computed by a constant size circuit.

**(Refer Slide Time: 09:18)**



above it.


↓

This dependence can be captured by a constant sized circuit.

Also, at the end we need to check if there is a q<sub>acc</sub> in the last row.

The size of this whole circuit will be  $n^k \times n^k \times O(1)$ .

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




So, it is constant multiplied by  $n$  power  $2k$  which gives us order  $n$  power  $2k$  sized circuit family to decide this language. So, in fact the major the bulk of the work is will be done by this kind of and this constant size circuits and its copies. But then one also has to verify that the start is a valid start and whether there is an accept in the bottom row. So, we have to also check whether there is an accept. So, that will be another small circuit family for that is there an accept state  $q_{acc}$  accept here.

That is also it is a very simple Boolean formula that one could or formula that one could construct. So, this whole circuit will take  $n$  power  $k$  times  $n$  power  $k$  that is  $n$  power  $2k$  cells multiplied by a constant size circuit which is exactly dealing with this computation this computation of the blue cell from the red cells. So, it is  $n$  power  $k$  times  $n$  power  $k$  times a constant which is  $n$  power  $2k$ . So, we have shown that given any language it has a deterministic any language in  $P$ .

That has a deterministic Turing machine that runs in some  $n$  power  $k$  time for some constant  $k$ . And correspondingly we get an  $n$  power  $2k$  size circuit family to decide the same language and which is what we wanted. So, now we have shown that  $L$  has a polynomial size circuit. So, any language in  $P$  is contained in  $P$  by poly. So, one natural question that one would ask is  $P$  is content in  $P$  by poly well is  $P$  equal to  $P$  by poly.

Because that is what one reasonable thing to guess so the answer is no, this is not the case  $P$  is not equal to  $P$  by poly mainly because this boils down to the non-uniformity. So, what do I mean by non-uniformity? Each language is captured by a circuit family not a single circuit and it can have different circuits for or it will have different circuits for different input lines. And this really makes a difference in the case of when you compare Turing machine based computation and the circuit based computation. So, circuit based computation is non-uniform and we will see the difference.

(Refer Slide Time: 11:55)



So is  $P = P/poly$ ? NO!  
X


---

Theorem: Suppose  $L$  is a unary language.  
(that is  $L \subseteq 1^*$ ). Then  $L \in P/poly$ .

Proof:  $L \subseteq \{1^n \mid n \in \mathbb{N}\}$ .

For each  $k$ , check if  $1^k \in L$  or not.

$$C_k = \begin{cases} \bigwedge_{i=1}^k x_i & \text{if } 1^k \in L \\ \perp & \text{otherwise} \end{cases}$$



So, we have a proof for that that  $P$  is not equal to  $P$  by poly so this is not the case. So, in fact we will show that there are undecidable languages in  $P$  by poly. So, let us see why, so first we will show that suppose  $L$  is a unary language right which means  $L$  contains only strings that are that are composed entirely of bonds not all the strings that contain that are composed of ones but some of these things. So,  $L$  could be 1, 1111, 11111, 10 1s maybe 20 1s are there and so on.

So,  $L$  is a unary language in that case we show that  $L$  is in  $P$  by poly. Why is this; the case this may not be that difficult for you to imagine because if  $L$  is a unary language. We already which means for each length  $L$  can possibly contain for each length  $n$ . So, let us say for length 4 so 1 1 1 1 is a is the string. So, either 1 1 1 1 is there in  $L$  or is not there in  $L$ . And if there is a string of length 4 in  $L$  this is the only string that is possible. So, either 1 1 1 1 is there in  $L$  or not there in  $L$ .

And similarly for the length 5, 1 1 1 1 1 is it either it is there in L or not there in L. And because there is only one candidate string per length it is not really that difficult to construct a circuit family. So, suppose L contains only strings of this type only comprising of ones. So, now let us construct the circuits for each k. So, for each k we need to check if one the string of length k the only string of length k that is this one power k is it in L or not.

If it is in L so recall for each k, we get to for each input length we get to make a circuit. So, if 1 power k is in L then the circuit for that k will be an AND of all the input bits. So, I could just have an AND of all the input bits and this is a polynomial sized circuit even if you are restricted to fan in 2. This is a polynomial size circuit which will only allow one power k to be in 1. If 1 power k is not in L that means there is no string of length k in L which means the circuit is just always going to output 0. So, again that is a very trivial circuit.

**(Refer Slide Time: 15:00)**

The slide contains the following handwritten text:

Theorem: Suppose  $L$  is a unary language.  
 (That is  $L \subseteq 1^*$ ). Then  $L \in P/poly$ .

Proof:  $L \subseteq \{1^n \mid n \in \mathbb{N}\}$ .

For each  $k$ , check if  $1^k \in L$  or not.

$$C_k = \begin{cases} \bigwedge_{i=1}^k x_i & \text{if } 1^k \in L \\ 0 & \text{otherwise} \end{cases}$$

The above family  $\{C_k\}_{k=1, \dots}$  decides  $L$ .

On the right side of the slide, there are two rows of four vertical bars representing the string 1111.

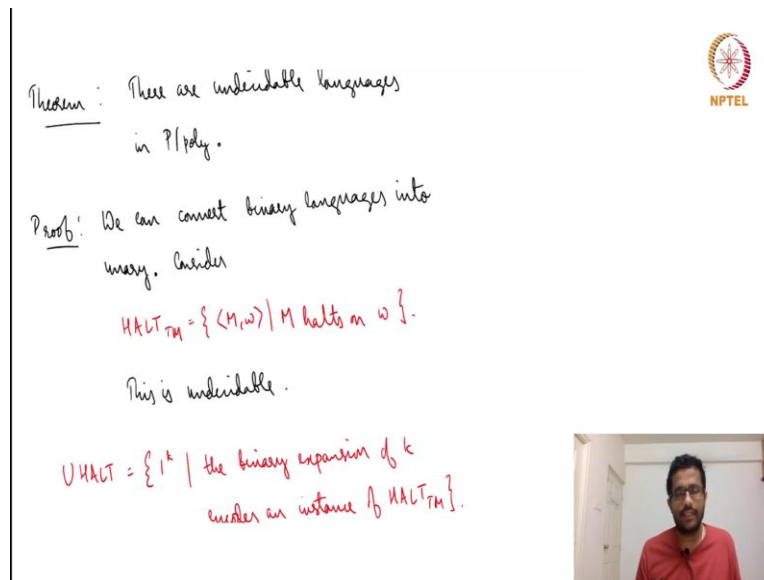
The NPTEL logo is in the top right corner. A small video inset in the bottom right shows the lecturer.

So, the circuit family is very simple for each k if k if one power k is in L then you have an and if 1 power k is not in L you have a zero and that is it. And this family is a family is a polynomial size family that decides L. So, what we have shown is that any unary language has a polynomial size circuit family, so any unary language is contained in P by poly. So, the real thing is that the non-uniformity combined with the fact that there is only one candidate string per input length.



Means that you could just look at the input length and say whether it is there or not that is all that there is. If it is a binary language then this is not this may not be possible because one has to look at the string and there could be multiple strings, multiple candidate strings per input length. So, you would need a more non-trivial circuit not such a simple circuit.

**(Refer Slide Time: 15:58)**



Theorem: There are undecidable languages in  $P/poly$ .

Proof: We can convert binary languages into unary. Consider

$$HALT_M = \{ \langle M, w \rangle \mid M \text{ halts on } w \}.$$

This is undecidable.

$$UHALT = \{ 1^k \mid \text{the binary expansion of } k \text{ encodes an instance of } HALT_M \}.$$

Now let us see why there are undecidable languages in  $P$  by  $poly$  and this will make use of the fact that all the unary languages are in  $P$  by  $poly$ . So, consider any undecidable language so maybe the favourite undecidable language is maybe halting problem. So, that considers the halting problem, halting problem consists of a Turing machine  $M$  and a string  $W$  such that  $M$  halts on  $W$ . So, given  $M$  and  $W$  one has to decide whether  $M$  halts on  $W$  or not.

And as you know this is an undecidable language, so there is no algorithmic process by which one can decide if a given Turing machine holds on a given input string. So, this is an undecidable language. So,  $M$  and  $W$  could have a binary description, now it is not such a big deal one could given a binary instance of a certain problem one could always convert it into a unary instance. So, given so you look at the description of  $M$  and  $W$ , it may be some number 0 1 1 0 something some long string.

So, now this is a binary number you could read it out as a binary number and if this binary number is a halting problem is in halt  $T_n$  then you include it. So, suppose this is  $k$  suppose is a  $S$

instance of the halting problem then you include it in a unary language, where include 1 power k in a unary language. So, if the binary representation of k is in halt, then you include the unary representation of k in unary halt. So, U halt stands for unary halt.

So, this is unary version of halting problem. So, the only thing that changes is from binary U come to unary. So, you can easily convert an instance of any language into a unary instance, but when you convert a unary instance what we said earlier applies that is for any unary language there is a polynomial size circuit family. So, unary halt is an equivalent representation of the binary halt but just with different input string lengths.

**(Refer Slide Time: 18:49)**

$HALT_{TM}$   $\{ \dots \}$   
 This is undecidable.  $0110111\dots 010101$   
 $k$   
 $UHALT = \{ 1^k \mid \text{the binary expansion of } k \text{ encodes an instance of } HALT_{TM} \}$   
 This is also undecidable, but unary.  
 So  $UHALT \in P/poly$ .  


---

 So  $P \neq P/poly$ .

And this has a polynomial size circuit family by above theorem, the previous theorem or by the previous theorem. However, it is still undecidable just because it is written in unary does not mean that you will get a language, you will get an algorithm for it or a decider for it. Because if you had a decider for it then you can always convert binary instances of halt into unary and then run the Turing machine.

So, it is still undecidable unary halt is just another representation of the binary halt but it is undecidable. So, it is undecidable but still has a polynomial size circuit family. So, what really is happening here is that even though it is undecidable when you make converted to unary now in

the case of binary there could be multiple instances of a certain length which is hard to decide. But in the case of unary all the inputs are spread out.

Because you only have one candidate input, one candidate per input length which may or may not be there and you could directly program it into the description of the circuit family. So, this means that there are undecidable languages in P by poly and this means that P by poly is not equal to P because P means languages that are decidable in polynomial time. Forget P it also shows that NP is also not equal to P by poly exponential time is also not equal to P by poly P space is not also not equal to P by poly.

Because all these things are decidable and but then P by poly has undecidable languages in it. So, P is not equal to P by poly.

**(Refer Slide Time: 20:49)**

So  $P \neq P/poly$ .

Decidable

$P$

$P/poly$

Where does NP feature in this?

Is  $NP \subseteq P/poly$ ?

So, just as a depiction the red circle denotes decidable languages and the black circle denotes P. Then P by poly contains P but also contains some undecidable languages, so it is just going out of the decidable circle. Now one question that somebody that is natural to ask is where does NP sit in all this, where it is sigma 2 set or NP set. So, is NP contained in this contained in P by poly does it do the languages in NP, do they always have polynomial size circuits.

So, this is a valid question to ask is NP contained in P by poly or is NP going to be something like this, is it some of languages in NP have polynomial size circuits but some others do not. So, this is a valid question that one could ask. So, that is an interesting question and we do not have a yes or no answer as of now like many for many of these questions. We do not know if  $P = NP$  also. But it has been shown that if NP were contained in P by poly like this

If situation was like this, then in that case there will be collapse of the entire polynomial hierarchy to  $\Sigma_2$ . So, because it is widely believed that  $P$  is not equal to NP and that the polynomial hierarchy does not collapse. It is also one tends to take this as an evidence that NP cannot be contained in P by poly or in other words it is believed that NP there are languages in NP that do not have polynomial size circuits.

Because there is some evidence that something else will happen that is that is unlikely. So, that theorem is called the Karp Lipton theorem and we will see it in the next lecture. So, in this lecture what we saw was that? We define P by poly which is the class of languages that have polynomial size circuits. We saw that deterministic polynomial time P is contained in P by poly. However, we saw that P by poly also contains undecidable languages hence P cannot be equal to P by poly.

So, just proper subset and we concluded by discussing about where does NP fit in all this. That is all from lecture 37.