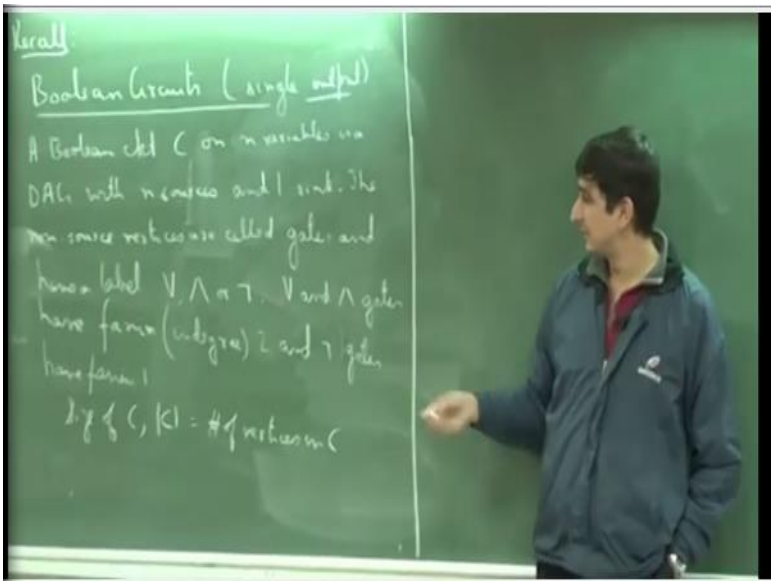
Computation Complexity Theory
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture-14
Boolean Circuits

(Refer Slide Time: 00:14)



So, let me just write down some of the things we did yesterday. Looking at this definition of boolean circuits. So, let us carry on with our discussion of circuit complexity, so we began yesterday by discussing the non-uniform mode. So, non-uniform model of computation where instead of requiring just one machine to be able to compute let us say some function or something for all strings in a language.

Let us say we have different we have a family of machines or in this case we have a family of circuits which computes a language. So, how do we model such classes and what does that give us? So, so we will assume for the time being that our circuits are single output circuits, that is they have one sync node and they have n sources and we will assume that we are looking at inputs of size n.

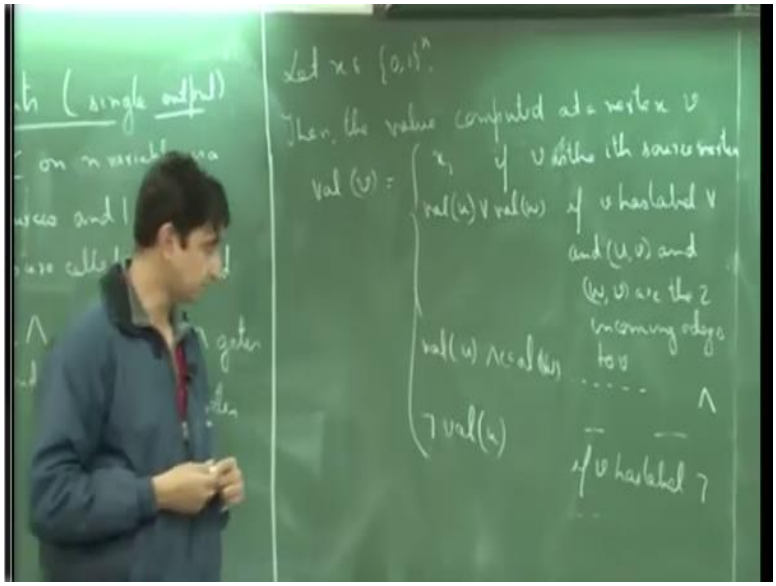
So, again for the time being we will just assume that there are no constants involved. So, yesterday I did say that you can have constants, you can still have constants but what did I say.

So, the n sources that you have, so let us just stick to the notation that these sources correspond to the n bits of your input string. Yes, it is not difficult to generalize this, but let us just have this for the timing.

And then the size of the circuit is basically the size of the graph, in the sense that how many vertices the graph has. So, how do we compute a value again that is done in a very in the natural manner, but I can mention it. Yes, so when I say vertices I mean sources as well. So, let us include sources as well because I was so basically the book has this definition.

So, yesterday I was talking about the sources as being input gates also. So, then you can talk about the number of gates but the way the book defines this a circuit is, they only call the non source vertices as gates, so that is why I want all the vertices to be part of the size. But actually as we shall see all these things do not matter much I mean there is only a constant factor difference between these things.

(Refer Slide Time: 06:22)

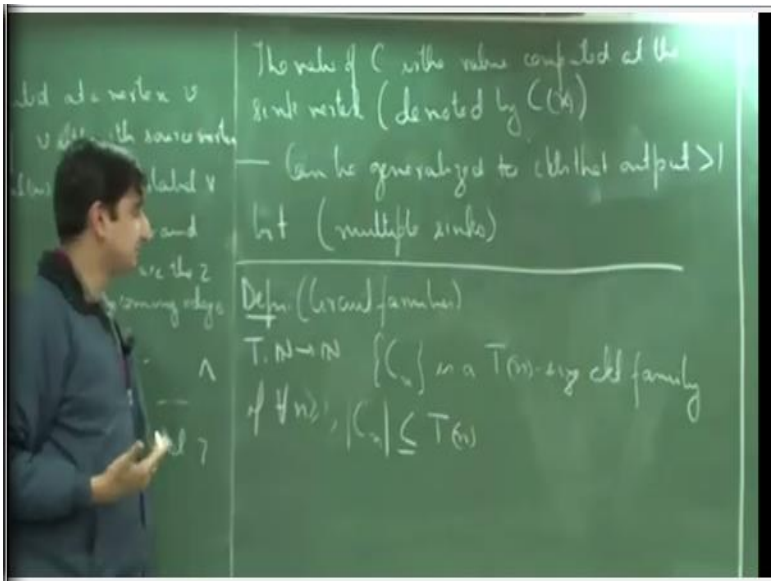


So, what is the value, so let x be some string of length n, so then the value computed at a vertex V will denote it by value of V, so this is equal to x i if V is the i th source vertex. So, we can assume there is an ordering among the source vertices and the value of the i th source vertex is the i th bit of the string, so that is why we just assume that there are no constants for the time being.

So, value of V is equal to some value of u or value of w if V has a label or and u v and w v are the 2 incoming edges to V, so just one more thing that I did not mention yesterday. So, we generally referred to the n degree of gates as the fanin of that gate. So, that is again fancy term but in circuit complexity you might often come across this, so just I wanted to mention that.

And similarly for AND and NOT also it is defined in the same manner it is value of u and value on w if V has a label AND. And these are the 2 incoming edges to V and it is not of value of u if V has label not AND. So, in this case it has fanin 1, so u v is the only incoming edge. So, this is how the value is computed at each vertex given a string x.

(Refer Slide Time: 09:40)

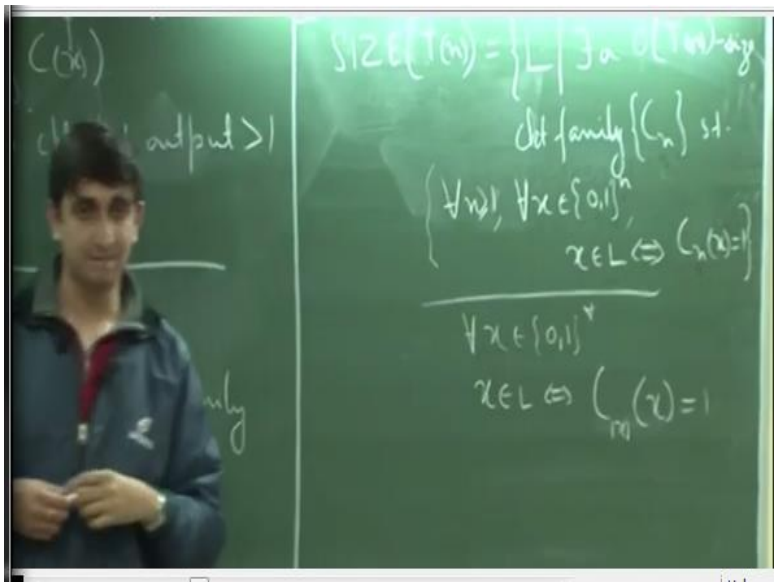


And the value of C is the value computed at the sink. So, this is the proper definition of a circuit and how does it compute a value. And now you can see that it is easy to generalize these 2 circuits which compute more than one bit of output. So, greater than 1 bit, so basically it will have multiple sinks, so this is the definition. So, now let us look at the basic class of languages that we can define.

So, suppose we have some function from positive integers to positive integers. So, we say that C n is a T n size circuit family, if for all n size of that circuit is less than or equal to T of n. So, basically this allows us to look at a circuit family of a certain size, so for every n I want that

particular circuit to have size no bigger than T of n. Now we can define languages also based on this idea of size.

(Refer Slide Time: 12:35)



So, let us call this class size T of n, so and as you must already be guessing what the definition will be, it is the class of all those languages L such that there exist a. Let me use the order notation here order T n size circuit family C n. Such that for all x of length n x is in L if and only if C n of x outputs 1, so I will just write it here, this is denoted by C of x. So, when I say C of x it is basically the value computed by that circuit on that input.

So, this size T n basically is the class of all those languages for which there is some circuit family of order T n size which computes every instance of length n in an appropriate manner. So, if that instance belong to that language then it will output 1 otherwise it will output 0, and this happens for all n. So, so this is the standard way of defining this, the only thing is that now we have a circuit family.

So, note that when we were defining things like d time T of n and n time and d space, what we were looking for is there exist some turing machine a turing machine. But here the crucial difference is that no longer are we restricting ourselves , so we are no longer restricting ourselves to just one machine or one circuit, we have a infinite family of such computational objects, that is able to decide the language.

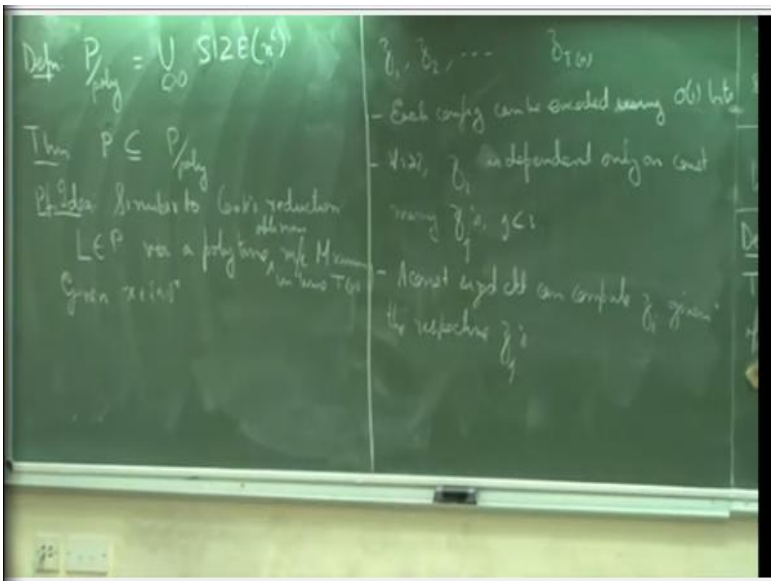
So, I should have written here that for all n greater than or equal to I can put 0 also. So, you have this circuit family n, so there are an infinite number of circuits, so for all n if you look at any string of that length, so that string will be in the language if and only if that particular circuit. So, the circuit corresponding to that n outputs 1 on that input for a particular n then so u also.

Yeah, so that is one way of doing it, so here, so that is also a good instead of writing n here I can put for all x in 0, 1 star x is in L if and only if C mod x of x = 1. I mean it is the same way of I am just rephrasing what I wrote here. So, instead of saying this what we can say is for all x when do I say that x is in L according to this definition. If that particular circuit outputs 1, so what is that particular circuit in this case?

It is the circuit C n where x has length n, so I am looking at all strings x of length n. Because if you look at the definition of a circuit by it is very nature it can only take those strings which have a certain length and nothing else. So, that is why it is in L if and only if C mod x on x = 1. So, this is just an alternative way of saying the same. So, now let us come back to the question that we began with yes. Left side n and size of T of n so let us, so I was little hesitant when I wrote 0, so let me put it as 1.

I mean I wrote 0 because I did not want to exclude the empty string, but that is just 1 string I mean that can be taken care of separately, I mean I can always start with n = 1, just some function over here n is just a dummy I mean here so maybe I should mention. So, here basically n just means that T is a function, so this n does not play any role with the n sitting inside, so that is so it is kind of a misnomer. But if I just write size of T, it does not quite say what T is, so that is why we write T of n yeah.

(Refer Slide Time: 18:40)



So, now we can define circuit families which have polynomial size, and that is written as P slash poly and it is equal to union over all C greater than 0 size of n to the power C. So, when I say n to the power C here this is the function that I mean that. I am looking at all those languages which have a circuit family that is no larger than n to the power C for every n and I am taking that union over all constants, so that is this class P slash poly.

So, what I was saying a little while earlier, so recall the motivation that I mentioned yesterday when we started our discussion that we are not able to show whether P is not equal to n P. But what about if we relax this question, what if instead of asking there is a some n P language which is not computable by a polynomial time turing machine?

What if we relax and ask that can we at least show that there is a family of such circuits which is not able to compute some n p language? So, this is the class which I was talking about, is family of two well you actually want more, you not only want a family of turing machines each of which is designed for a particular input, you want all those turing machines to run in polynomial time as well, so yeah so this statement is actually equivalent to that.

And actually we will actually come to that little bit later today but maybe I can mention it since you raised it. Actually that is not very difficult, so suppose if you have a family of circuits which decides a certain language it is very easy to convert that into a family of turing machines again

with decide the same language. Because what you can do is you can just design the transition function of the turing machine in such a way, so that it simulates that circuit.

So, whatever that circuit does the transition function of the turing machine just copies that behavior. If you are computing, you have your input if you are computing and of two bits your transition function will accordingly compute the AND of those 2 corresponding bits and or not whatever, so that can very easily be done. So, that is why but the good thing about circuits is that there is much more structure into a circuit.

I mean see if you look at a turing machine it is more like a black box sitting out there, I mean there is one turing machine which given an input will tell us yes or no. I mean of course there is the transition function that we have but well we do not know much about the transition function as well. But on the other hand if you have a circuit, so this has a much richer mathematical structure embedded on it.

We can actually say more stronger mathematical statements which it is not so easy in the case of turing machines. So, I will mention this thing in more detail while we maybe today or tomorrow but let us see. So, before I precede any questions not really right because I mean if you have a circuit what do you mean by the time taken by that circuit.

So, depth basically means that what is the longest path from a source to a sink note. So, if you take the maximum overall such shortest paths that will give you the depth of a circuit and that is also a concept which people study and we will also look at it in a couple of days time. But generally it is size and depth which is so these are the 2 most important parameters that people study about circuits.

So, the first thing that we will show and this is also not quite this is again very easy to show that. P is contained in P slash poly, so in other words if you have a language for which there is a polynomial time turing machine, you can design a family of circuits each having polynomial size which will accept that same language. And I will just give the proof idea because the actual proof we have already seen something similar earlier.

So, the idea of this proof is again similar to Cook's reduction ok, so where we showed that sat is n P complete, actually sat is n P hard. So, it is basically the same idea that is being used to prove this. So, let us quickly go through the basic steps of this proof actually some things are important to note. So, what we do is, so suppose we have a language L in P which is accepted via a polynomial time during machine M.

So, now I want to design a family of circuits, so given any string x, so if you have any string x let us look at the configurations that this machine will take the various configurations that this machine will take on this input x. So, what can we say about let us say the configuration graph of M on x, does it have any particular structure, yes it is exactly, so it is just one path because from every configuration you can go to exactly one other configuration.

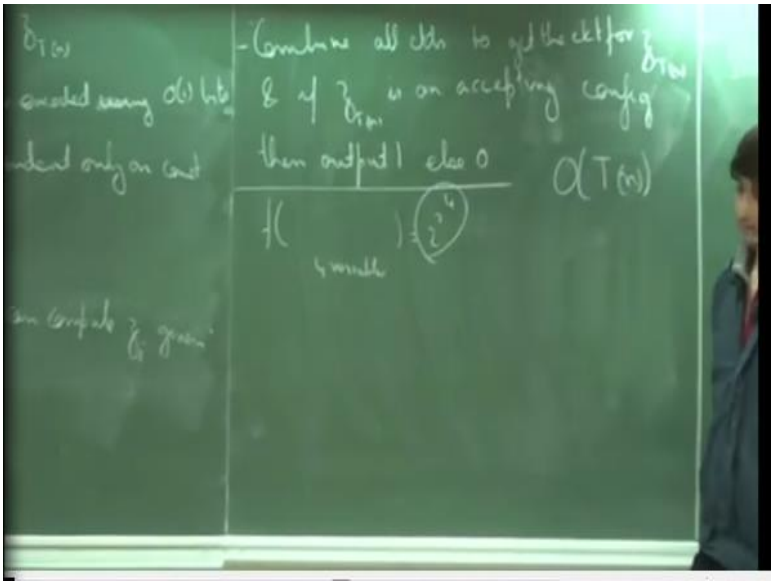
So, **so** let us write down the configuration. So, let us say this is my starting configuration, this is the next configuration and so on the final configuration is some T of n. So, poly time machine M, running in time T of n. And we would also assume one more thing about this machine, so we will assume that this is an oblivious turing machine. Say, in other words the computation of the machine does not depend on the actual input.

So, it only depends on the position of it is input head, so these are my configurations. So, now note the following things about these configurations. So, each configuration can be encoded using some constant number of bits. Because what is the configuration? It is basically a state and the content of it is various heads. So, that can be encoded using some constant number of bits.

And for all i greater than or equal to 2 z i is dependent, so only on constantly many z j's where j is less than i. So, there are only constantly many such configurations on which it will depend. In particular of course it will be dependent on the previous configuration and also it will be dependent on that configuration where the input head was last scanning that same position. If it has only one work tape then that is the only thing, and of course it will depend on the input head position also.

So, now we have these things, so each configuration has a constant size encoding and each z i can be is dependent only constantly many z i's. So, therefore a constant sized circuit can compute z i given the respective z j's. So, suppose you are given whatever are those respective circuits for all those previous z j's there is just a constant size circuit which will compute the value of z i for you. So, now it is basically just combining all these circuits in the proper manner.

(Refer Slide Time: 30:22)



So, combine all circuits to get the circuit for z T of n and if z T of n is an accepting configuration. Then accept well I should not say accept then they I should say an output 1 else 0. But what is crucial here is that we are only looking at an input of a particular size, so that is where we are using the fact that we have an oblivious machine. So, for different input lengths you will come up with a different circuit.

Yeah, so given x belongs to, so I should say that as well. So, we know that z i is dependent only on some constantly many z j's so suppose we assume that we have circuits for all those z j's. So, there are circuits which are able to compute that. So, since this z i only depend on what those constantly many z j's are and to represent each configuration we need only constantly many bits. So, therefore given these values we can always decide that by using a constant size circuit.

Because the number of inputs to this circuit is constant and that basically forces this also to have constant size because otherwise you will be repeating some things, no that I am not saying in one

step. So, that assumption is there and so the reason for that assumption is what I am saying. So, if you look at one step of a turing machine, suppose we are looking at just one step, so what is happening? Basically I am going to some z i - 1 to some z i and this z i depends on some constantly many earlier configurations, so let us say depends on three other configurations, depends on z i - 1 and maybe some two other configurations.

So, basically what is the transition function now, the transition function is a function which takes 3 arguments. I mean one way to model the transition function here is, if I want to compute what the configuration z i is it is just a function of three other configurations. Now what I am claiming is and also what we know is that each configuration can be encoded using constantly many bits. So, now what I am claiming is that suppose somebody provides me with the values of those three arguments, what those three arguments will be?

And here I am using the fact that they have constantly many bits. So, now the circuit size that I need to compute z i from those constantly many bits cannot be larger than constant. Because if it is large I mean if it is larger than some constant then basically you can use pigeonhole principle or something like that, to show that you are performing redundant information because if you have let us say a function let me tell you a simpler thing.

So, suppose if you have a function on 4 variables, so how many different functions are possible on 4 variables? Binary functions yeah not 2 raised to the point say it is 2 raised to the power 2 raise to the power 4. Because for each setting of these four variables, so there are 2 to the power settings of these 4 variables and for each setting it can take either a value 0 or 1. So, now suppose you relax this to any array k function where the function outputs a string having k bits, so it will be some k to the power 2 to the power 4 something like that.

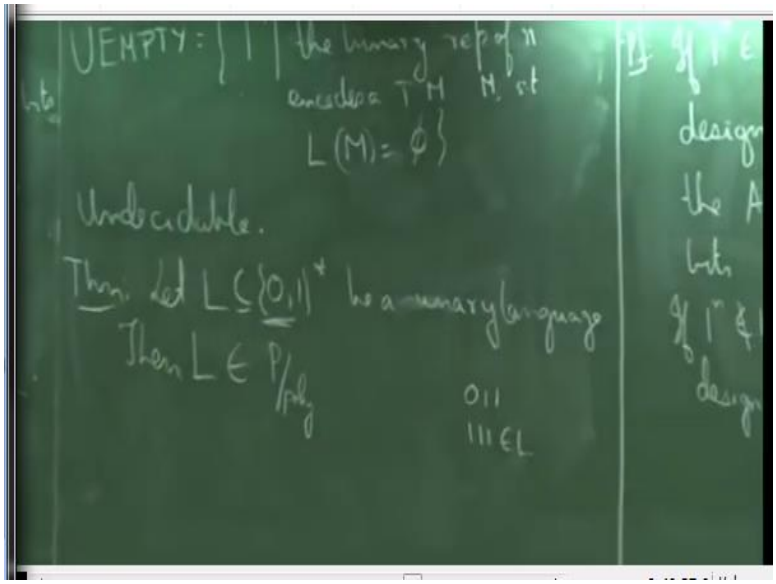
But the point is that if this is constant and if that k is constant then this has to be a constant. So, the number of functions itself is constant, so therefore the size of that circuit which computes that will be constant, so that is the implicit nature. But the size basically blows up when you combine all these things. So, here we are assuming that we are already provided with these z j’s when we are computing z i's but how do we get those z j's.

There is the linear chain which gets built up and that is why when you combine all these circuits to get the final circuit, so this will have some size some order of T of n, because there are T of n configurations. So, the essential idea behind this and also the Cook's reduction is that you can very easily go back and forth between this circuit model or let us say formula whatever and the turing machine model.

So, if you have a circuit you can always simulate it on a turing machine, and if you have a turing machine that always there is always a corresponding circuit for computing the same thing. The only difference is that since turing machine is a uniform model to go to a circuit model you will need a family of circuits, so that is the only difference. So, any other questions? So, we will look at that, no P by poly does it include n P no, so that is not known, so whether P by poly includes n P as well, so that is not known.

In fact that is not believed to be true as well, so that is what we will probably show tomorrow. But actually it is a class which is incomparable with all these classes P, well it is comparable with P but it is incomparable with all the other super classes of P, such as n P, P space, e x P and that will see just now why it is incomparable? It is a very easy argument for that oh I erased this, but so I hope everybody was done copying this part of them.

(Refer Slide Time: 39:17)



So, let us look at the following language let us call this UEMPTY, so UEMPTY is the unary encodings of all those machines n. Such that n the binary representation of n encodes a turing machine m such that L of M is empty. So, those are not the s instances, so I will look at all those n's which are valid encodings of turing machines and their languages phi, the language of that machine is phi.

So, if you have an n which is not even a valid encoding of a turing machine, then you do not include that in this language. So, what can we say about this language, so again this is from a TOC course point of view. Say is this language a decidable, so this is undecidable, and why is it so? Anything you can prove it using Rice's theorem or whatever.

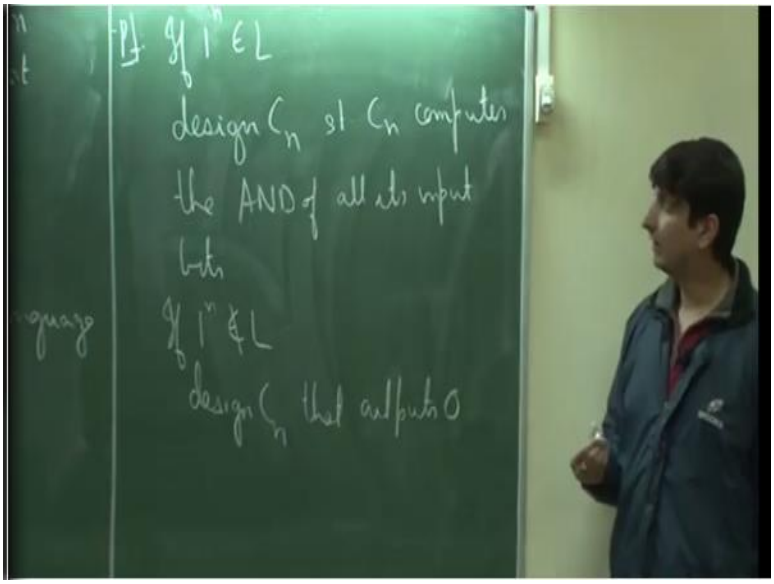
So, at least if let us look at the following language, so suppose if you have the binary representation of a turing machine M such that L of M is empty that language is undecidable. So, that probably you have seen in your TOC course or even if you have not seen it is very easy to prove, I mean it is very easy to reduce let us say the halting problem or any other undecidable problem to that problem, if you have the binary representation.

What I am claiming here is that even the unary representation is undecidable because suppose not, suppose this were to be decidable what I can do is? Given a binary representation of a machine M I can very easily convert it to it is unary representation. I mean I can easily look at a binary number and convert it to a unary number, and then I run the machine for UEMPTY and that will allow me to decide the other language, and that we know is not possible.

So, therefore this language is also undecidable. But what can be shown is the following is that suppose you have any language L, so maybe it is over binary symbols but L is a unary language. So, then L actually belongs to this class, so no matter what unary language you pick that always belongs to P slash poly. So, again the proof is actually very simple, so any idea how we can prove this is?

Basically that is you have to be little careful in the accepting part, but basically that is the idea. So, now note that all we want is a family of circuits, so that is for each length n, I want a circuit which will either accept or reject that input.

(Refer Slide Time: 44:16)



So, what I will do is if 1 to the power n belongs to this language L, so then design C of n such that C of n computes the AND of all it is input bits. And if 1 to the power does not belong to n then design C of n that no that is not, you mean for this, I mean you do not need I mean you just, I mean you can just design a very trivial circuit which will always output 0, you do not even need to look at the input, so such that output 0.

Now because of this know, because I am assuming that the language can be over binary strings. So, I can have a x which has some zeros also in it. So, maybe I have an x which looks like this 010 sorry 011 and also the string 111 is in this language. So, I do not want to accept this but I want to accept this, so that is the reason. Because it is, so it is an it is an unary language, so it is sufficient to take the end of all the input 2 over 1 yeah that is true.

Alphabet is binary, actually it does not matter I mean I think you can also assume the alphabet to be unary because even then this language would be undecidable. So, the whole point is that there are undecidable languages which are there in P slash poly, so that is the whole point.

(Refer Slide Time: 47:04)

So, P slash poly contains undecidable languages. Yeah here, so suppose you look at, so suppose 1 to the power n is in L, in particular I am looking at n as having the value 3, so 111 belongs to L. But since the theorem is for L which is a unary language 011 is not in the language, so the circuit that I designed for C 3 that should reject 011 whereas it should accept 111 and that is the reason why I am taking the end of all bits.

So, even if it has 10 then it will output 0, only that will be accepted. But see this is kind of a cheating in some sense, right, because I am that is not the problem. I mean you can assume the alphabet to be unary here also and the same thing will also carryover. In which case if 1 to the power n is L, you just design a trivial circuit which outputs 1 and 0, that is not the point.

But the cheating is in the place when we are designing the circuit, so what does circuit family say by definition? So, we say that a language is accepted by some circuit family if there exists a circuit family. We never talk about how we get that circuit family or how to construct such a circuit family? So, we are just exploiting that freedom that is given to us by the definition in proving that an undecidable language is contained in P slash poly.

So, this is something, well I mean there can be cases where it is useful but again this is something which we would like to restrict. We would also like to look at those kind of circuit

families where the circuits are constructible in some reasonable amount of resources, so that will make I guess more sense in certain cases. I mean certainly it will rule out this kind of a behavior.

So, what do you mean so reduction, so if you look at polynomial time reductions of course P by poly is closed under polynomial time, it is closed under both polynomial time n log space reductions. You can look at turing reduction, but during reduction I am not sure if P by poly is closed under during reductions also. But I mean we have not seen turing reductions yet, so I will not discuss about them.

But I mean I doubt if there are any other powerful reductions under which P by poly is closed. Because even this is not known, so whether NP is contained in P by poly or not, so this is not known, so this is open. And in fact this is believed to be false actually, that is what we will discuss I guess tomorrow. That if this happens that if NP is actually contained in P slash poly then the polynomial hierarchy will collapse to sigma 2 P, that is the second level.

So, that is why this is not believed. So, I do not know if it is closed under any other family of reductions. So, it is uncomparable to NP certainly but it is it contains P well that is not again a very correct statement to make it does not contain each undecidable language. The unary version of each undecidable language is contained in P slash poly because if you look at the binary version of this language that we do not know but that is probably not true in fact.

In fact we look at something of that flavor I guess not today either tomorrow or in the next class where we will show the existence of functions which cannot be computed by circuits. So, there does exist functions boolean functions which cannot be computed by circuits. So, let us look at just one more thing and then we will stop today. So, I mean how can we circumvent this problem.

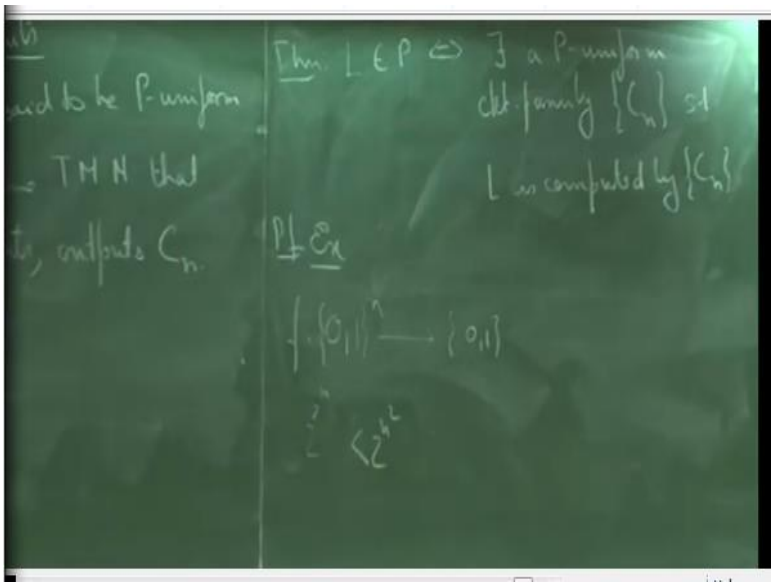
(Refer Slide Time: 52:22)

So, we look at or will define a sub class of these kinds of circuits known as P uniform circuits. So, what are P uniform circuits? So, a family of circuits C n is said to be P uniform, if there exist a polynomial time turing 1, n to be provided in unary. So, then what it would do is, it would output the corresponding circuit C of n. So, this P comes from the fact that this is a polynomial time turing machine.

So, now we have a certain degree of restriction on the families of circuits that we are talking about. So, suppose if you have a P uniform circuit family, so suppose if you have a P uniform circuit family which computes a certain language. Then what you can say is, given any input string x first you look at the length of that string. Suppose you are provided that string to a turing machine, just look at what that length is.

And then you simulate this machine M on 1 to the power n, so that will give you the circuit C of M. Now you can simulate that same input on the circuit and then you can answer yes or no. So, basically this is a more proper way of coming from languages which are accepted by circuits to turing machines. So, in fact I will just state that as a theorem , I will just leave the proof as an exercise because it is not very difficult, it is just combining the ideas that we have already seen.

(Refer Slide Time: 54:48)



So, a language L is in P if and only if there exists a P uniform circuit family C n such that L, I just write it in short, L is computed by this family. So, in other words if you look at any string of length n and if that string is n L then C of n would output 1 otherwise 0, the usual definition. So, I will just leave this proof as an exercise. So, suppose if it is NP coming up with a circuit family which s P uniform.

So, actually it is basically nothing it is just, so whatever the proof of that we gave earlier that P is contained in P slash poly, you can very easily figure out that it is a polynomial time construction. So, the circuit that we computed for that language that circuit is a polynomial time construction. So, that is the proof of the forward direction, and for the reverse direction it is just taking that circuit, taking an input, constructing that particular C n and simulating that input on that C of n.

So, I will just leave this as an exercise, so any questions? yes, not the power of the circuits, how the circuits are obtained in some sense. I mean it will restrict the family I mean the class of languages accepted by such circuits, yes. So, that is basically what this means know, so it is basically exactly those languages which are in P. So, not only the turing machine can compute, it can compute them in polynomial time.

So, this is kind of a (()) (57:38) restriction but we cannot help it, any other questions? sigma star yes, yes, no but this is a class of languages, so when you are talking about sigma star, sigma star

is one language, it contains another undecidable language as a subset. In fact all languages are subsets of sigma star, you are right but then decidability or any other such property is not closed under the operation of containment. So, that is a very easy thing to see, but here what when we are talking about P by poly this is not one language, this is a whole family of languages, it is a class of languages.

So, what I am saying is that this class of languages contains certain undecidable languages as well. So, it will be more than 2 to the power n probably, if they accept different functions. So, let us look at it representation I mean how do you represent a circuit. So, suppose you have a circuit on n variables. So, one way to so let us say it is a circuit of size n, so how do you represent a circuit? So, one way of representing circuits is basically by graph, and that is the most natural way of representing a circuit.

So, now the question therefore is that how many graphs are there on a n vertices, how many directed acyclic graphs are there? And actually you have certain other restrictions as well that is each vertex of that graph has in degree at most 2, it can have 1 or it can have 2 also. Yeah**,** but that will not, that is true but that will not reduce it significantly, but even then I mean even with all these assumptions actually you can show that it will have more than 2 to the power n graphs.

There can be 2 circuits which do exactly the same, yes, exactly the same way yes, so should we consider those 2. Let us say even if we consider them as the same. Then we can also look at the input and the output only because we are given a string and we just output 0 or 1. So, no but there is a size intermediate size know, so see the number of functions if you look at this, so that is it is not sufficient to look at that, because that is a huge number.

As I said little while earlier if you look at the number of such possible functions, so this number is much larger, so this is some 2 to the power n. But the number of circuits which can compute a function on n variables. So, that is probably I mean that is definitely not as large as this, but it is larger than 2 to the power n, it is definitely larger than 2 to the power n.

Because you cannot represent a graph using n bits, if you have an n vertex graph you need more than n bits to represent it. I mean even if you look at let us say an adjacency list representation, there are n rows in that thing but you need to have some non trivial graph it has I mean M is of the order of n again, because the degree is bounded. But it is of the order of n because degree is bounded.

So, size will be order M square, so this is clearly true. So, the number of circuit is not greater than 2 to the power n square, so it is certainly less than 2 to the power n square. And actually it is also less than probably some 2 to the power n log n or something like that. So, I do not know the exact bound on it, but it is greater than 2 to the power n definitely, that can be proven**,** so we will meet tomorrow then.