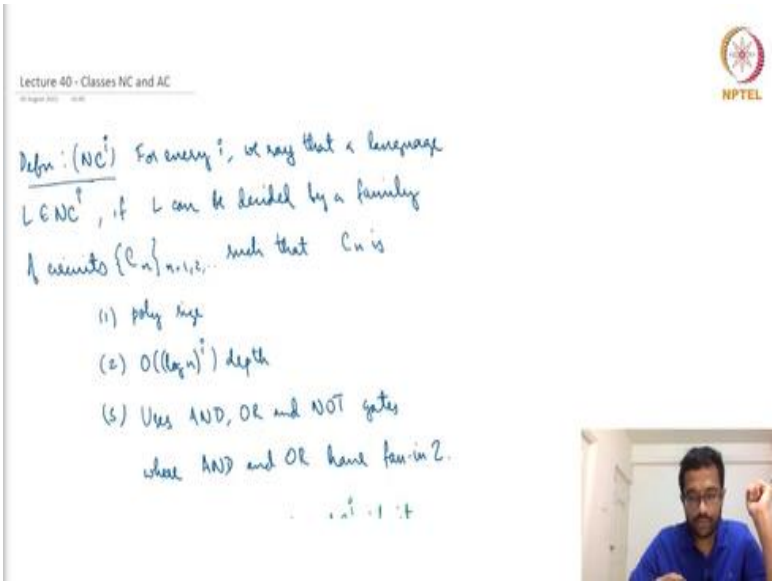
Computational Complexity
Prof. Subrahamanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture - 40
Classes NC and AC

(Refer Slide Time: 00:16)



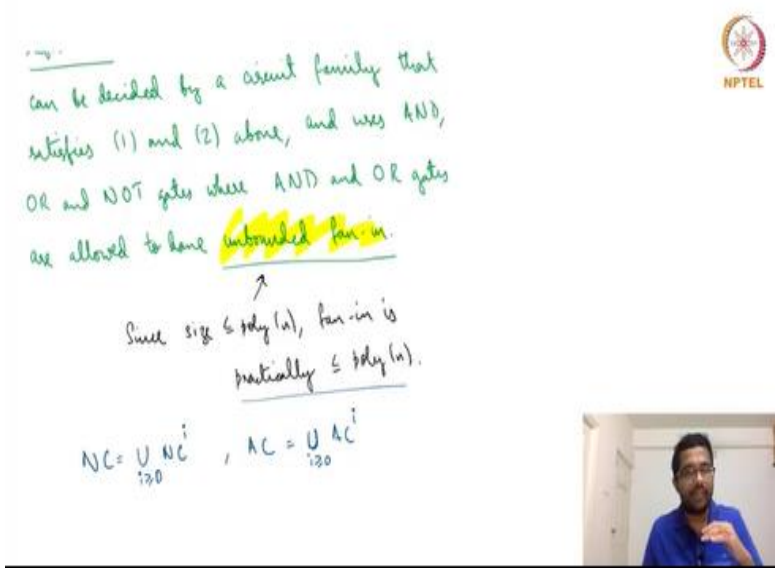
Hello and welcome to lecture 40 of the course computational complexity. So, far we have seen various circuit various complexity classes based on the circuit complexity such as P by poly and uniform variants of those and we have seen how they are relate to whatever we have already seen so far such as polynomial hierarchy for instance. We have also seen how much size is required to compute certain functions languages etcetera.

Today we are going to see two sets of circuit of classes called NC and AC which are actually both of which are contained in P by Poly but are more fine grained in the definition. So, let us define NC or NC superscript i. So, NC superscript i are this class of languages that can be decided by a family of circuits that have polynomial size, the circuits should have polynomial size and the depth should be log n power i and must use AND, OR and NOT gates.

So, far I think we have mostly been talking about circuits restricted to AND, OR and NOT gates but I am just restating it. And the most important thing is that these gates in the case of NC must

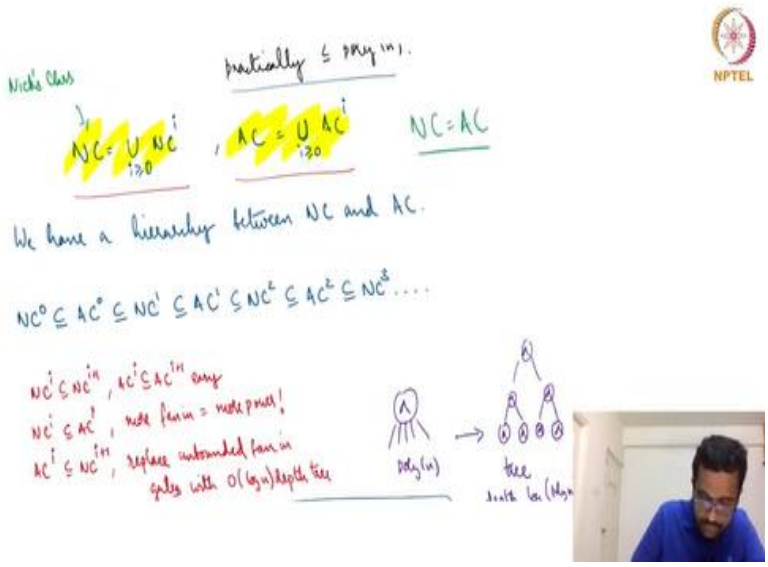
have fan in equal to at most 2. So, obviously NOT gates have finite one so and or cannot have fan in bigger than 2. So, this is NC i. So, NC i means when i increases you get more and more depth but all of which is limited to polynomial size circuits.

(Refer Slide Time: 02:04)



And AC i is a similarly defined class everything is the same except that the only difference being the AND gate and OR gate can have unbounded fan-in the and gate and or gate can have unbounded fan so this is the main difference. So, here they have fan-in 2 in the case of NC. But in the case of AC, they can have unbounded fan-in.

(Refer Slide Time: 02:33)



And unbounded really means it is enough to have a polynomial sized fan-in and because or find it to be polynomial in n because size is itself polynomial n. So, practically the fan-in could be polynomial. So, now let us see so this is AC i and NC i. So, both of which have polynomial size log n to the i power, log n power i depth and is constitutes of AND gate OR gate and NOT gate. In the case of NC i we said that the AND gate and OR gate cannot have fan in more than two.

But in the case of AC i they could have unbounded fan-in. And the class NC just without any superscript is just nothing but the union of all the classes NC i and similarly the class AC is the union of all the classes AC i. So, it is an infinite union. So, Nick Pippen ger used to work on these kind of circuits circuit classes a lot. So, NC was informally or the name NC was stood for or stands for Nick's class.

So, this was named by Steve Cook in the owner of Nick Pippen ger and in turn Nick Pippen ger named another class SC called and he called it Steve's class in the honour of Steve Cook. So, they did this kind of mutual friendship kind of fun thing during that time. So, AC stands AC does not stand for a name it stands for alternating class. So, because it is it corresponds to an alternating Turing machine class.

So, that does that is not have an interesting story like this Steve's class and next class. And we will not be seeing the Steve's class during this course. And actually, there is a hierarchy between all these NC classes and AC classes. So, let us see what is that. So, one thing that you may have already observed is that as you provide more depth you get more power. So, each NC i, NC 2 will obviously be contained in NC 3 which will be obviously contained in NC 4.

Because whatever can be done in NC 4 can certainly work under NC 3 and whatever can be done in NC 3 can certainly be run in NC 2. So, NC 0 is certainly contained in NC 1 which is continent NC 2 and so on. Same thing holds for AC as well so AC 0 is contained in AC 1 and so on. However how are these interrelated? The first thing that you may note is that NC and AC are exactly the same except that AC allows unbounded fan in.

So, this tells us that whatever can be done in AC i can certainly be done in NC i, sorry whatever can be done in NC i can certainly be an AC i. Because if you have unbounded fan in you are not bound to use it you could just use them as fan in two gates. So, NC i I will just write down NC i is contained in NC i + 1 this is easy and AC i is contained in AC i + 1 this is also easy NC i is contained in AC i.

Because more fan in equals to more power and finally AC i is actually contained in NC i + 1. So, this is probably the only interesting part. This is because AC, what does AC have that NC cannot have it is the unbounded fan in AND and OR. So, suppose AC had an suppose an AC circuit had an AND gate that took poly n fan in. What we can do is we could replace this gate we could replace that with something like this and we could replace that with a tree of sorts something like this.

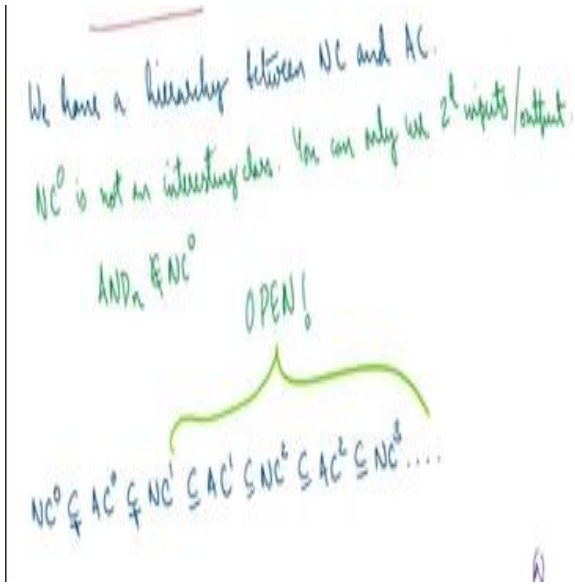
A tree or fan in two gates which will actually be computing the AND again. So, we could replace it with a tree and what would be the depth of this tree. So, the depth of this tree will be logarithmic of polynomial n which would be depth log of polynomial n which will be actually some order log n. So, because simply polynomial is upper bounded by some n power k. So, it is just k log n, so by replacing unbounded fan in gates with O log in depth tree.

So, basically the blow up of the depth so polynomial size it means polynomial sign because log n does not change. But the depth there is a depth blow up of log n. So, the AC i circuits have depth log n power i but then there is a further blow up of log n. So, log n power i multiplied by log n it is log n power i + 1. So, what we get is a circuit with polynomial size where the gates are AND, OR and NOT with fan in 2 but where depth has increased from log n power i to log n power i + 1.

So, it is NC i + 1. So, now summarizing all this we could just write it like this. So, we have NC 0 contained in AC 0 continent NC 1, AC 1 and so on, it is a chain. And since we have this nice relation, it also turns out that these two classes defined above NC and AC they are virtually the same thing. Because if it is an NC i then it is an AC i as well sorry if it is an NC i that is an AC i as well, if it is an AC i it is an NC i + 1.

So, when you take the infinite union, they are both the same. So, basically NC is actually equal to AC and this hierarchy one interesting point is that I want to mention is that NC 0 is not very interesting. So, maybe I will just make some space here and NC 0 is not very interesting.

(Refer Slide Time: 10:21)

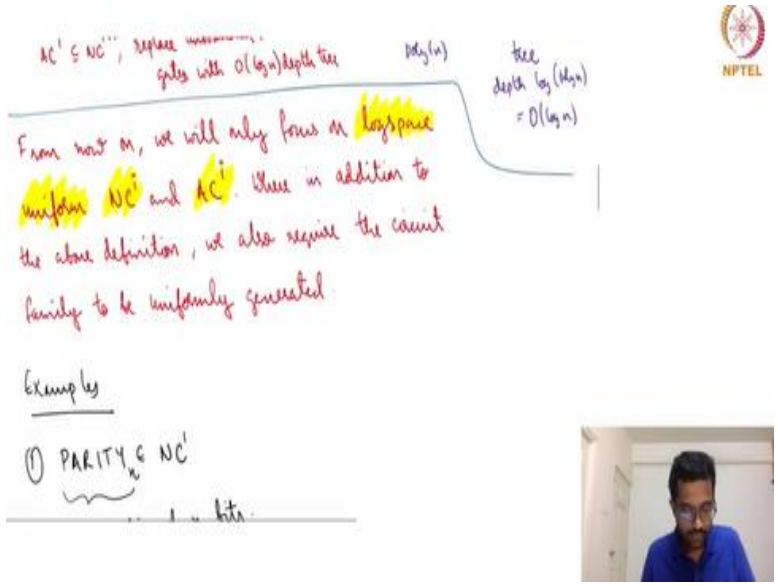


It is not an interesting class; this is because you are allowing polynomial size and constant depth alone with gates of fan in 2. So, you can only check you can only use two power d inputs for a certain output each output is a function of only two power d inputs. So, it is only looking at a constant number of inputs. So, this is not very interesting not much can be done in using this kind of setup. So, NC 0 is not very interesting.

For instance, even simple things like AND gate on n inputs is not in NC 0. But AND gate on inputs is certainly in AC 0 because it is you can just use that one gate alone. So, this inclusion is strict, NC 0 is a proper subset of AC 0. We also know that AC 0 is a proper subset of NC 1 because there are languages which we will see so parity for instance which is an AC 0 but not sorry it is an NC 1 but not in AC 0.

And what is interesting is that the rest of all the inclusions all these are open. We do not know if the remaining inclusions are strict or proper or otherwise or are there two classes that they are equal. So, this is something I want to say.

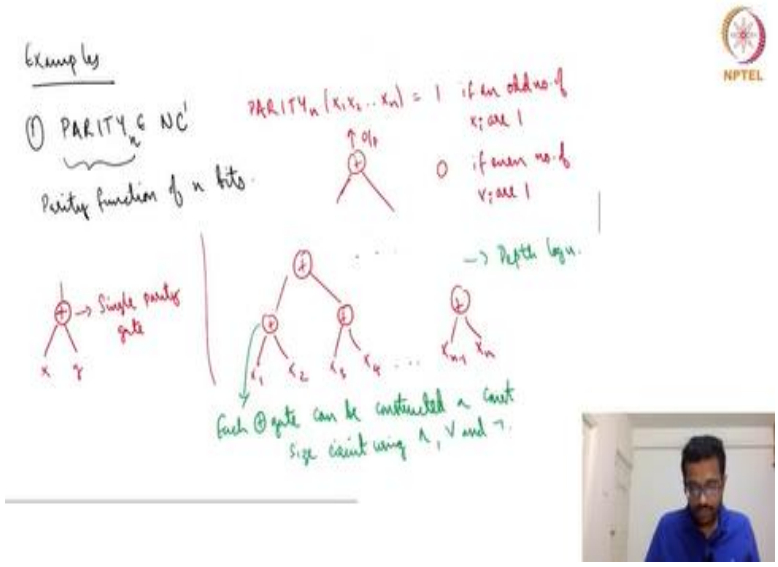
(Refer Slide Time: 12:25)



And I defined in the previous lecture you could also define log space uniform NC and AC. Whatever I said holds for all the NC and AC languages but now let us focus on log space uniform NC and AC. So, from now on we will just stick to log space uniform versions of NC and AC which means that in addition to what we have defined already. We require that the circuit family must be uniformly generated by a log space algorithm.

So, given input k in unary it should output there should be log space algorithm that generates a circuit for inputs of size k. So, now let us see some examples and after which we will see some. So, these examples will give us a general idea and after these examples we will see we will conclude and summarize.

(Refer Slide Time: 13:33)



So, the first example is parity. So, parity is just simply if you have n inputs it is a generalization of XOR. If you have n inputs in input bits parity is 1 so, parity of x 1, x 2 up to x n this is equal to 1 if an odd number of input bits are 1 odd number of x i are 1 and it is 0 if even number of x i are 1. So, that is parity and this is in NC 1 which means we are allowed login to the one depth. So, let us see why that is. So, notice that we could make a parity gate.

Let us say we make a parity gate and we denote it like this. So, this takes two inputs let us say x and y and outputs one if exactly one of them is 1 and 0 otherwise. So, this is a single parity gate and this can be constructed using a constant depth circuit you could just use and gate AND, OR gate to construct this. So, you can you can verify that. And now using this single parity gate you could build a parity n gate. So, you could have x 1, x 2, x 3, x 4 and so on, x n - 1 x n.

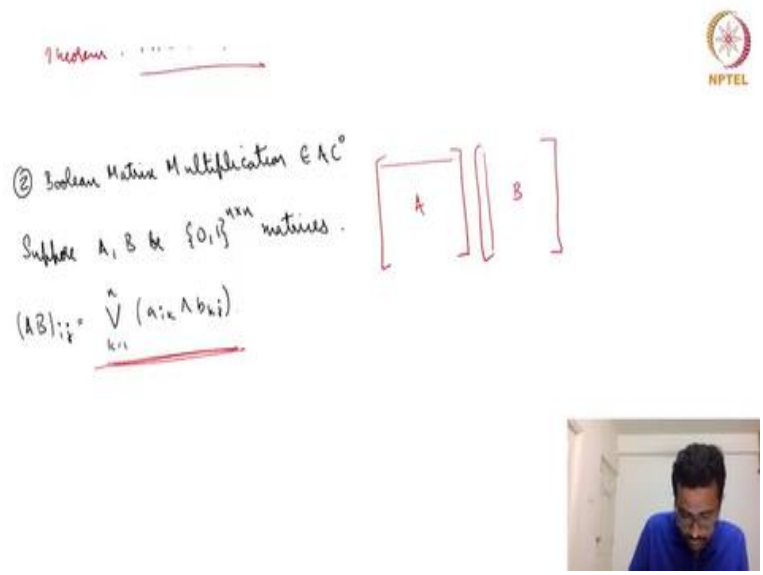
So, you could have some construction like this. So, finally culminating at the output. So, this has a depth order log n because each parity gate has a constant size. So, this the way I have drawn it is not a NC 1 representation because I have constructed using single parity gates. But then you can replace each single parity gate with a small circuit that uses only AND gate and OR gate of bounded fan in, you can work that out. So, this is certainly possible.

So, where I just perhaps write that each parity gate can be constructed by a constant size circuit using AND, OR and NOT. So, this is and now the depth if you measure there are n inputs the

depth is log n and this has depth log n. So, this is it and you can verify that this indeed computes the parity function. So, if you just cascade the two bit parities as a tree then you will get indeed the nth bit parity. So, parity is in NC 1.

And what is known which we will see later is that this was somewhat of an important theorem that parity is not in AC 0. So, this is an interesting and involved proof perhaps one of the longest that we will see. So, this shows that this containment here is strict, we have a language that is in NC 1 but not in AC 0.

(Refer Slide Time: 18:12)

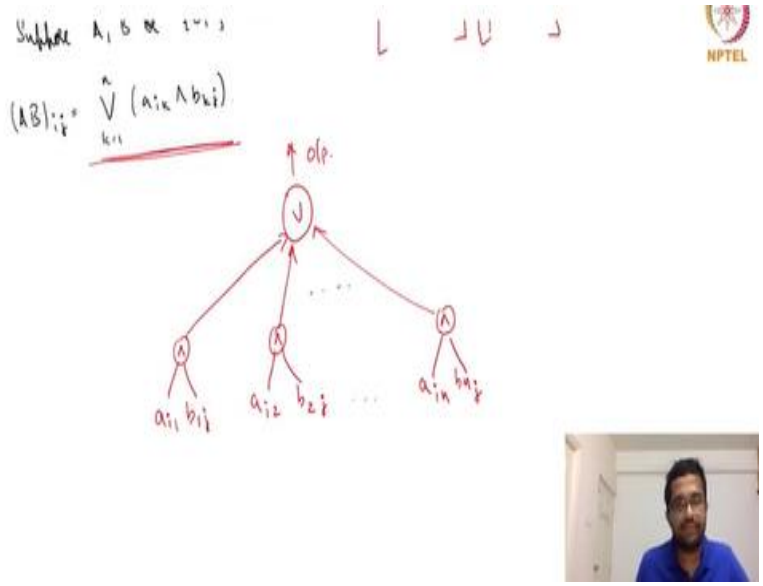


The next thing is a Boolean matrix multiplication and this is an AC 0. So, let us see what is Boolean matrix multiplication and why this is the case? So, suppose A and B are 0 1 matrices suppose this is A this is B n by n matrices as well 0 1 n by n matrices and what is Boolean matrix multiplication? So, in a standard matrix multiplication we would take the let us say one row of A and one column of B such that we will add up like A 1 1 will be multiplied with B 1 1 + A 1 2 will be multiplied with b 2 1 and so on.

So, what we will have in a usual multiplication is the i jth entry of A B will be summation of k = 1 to n of a ik multiplied by b kj. But Boolean matrix multiplication means we will replace summation with an OR and we will replace multiplication with AND. And do just to remove

confusion this, I just wrote this for explanation just to remove confusion in the notes, I will just erase this. So, this is the Boolean matrix multiplication.

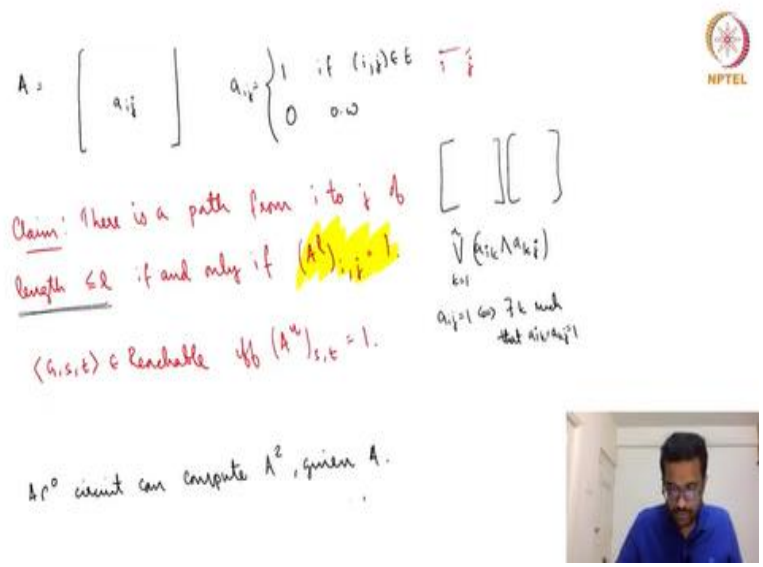
(Refer Slide Time: 19:40)



And we can do it using AC 0 which is a class of circuits with constant depth, AC 0 means constant depth but allowing unbounded fan in gates. So, why is that? So, I will just compute I will just show you how to compute ab, the product i jth of the product. So, we have a i1 and b 1j, we have a i2 and b 2j and so on up to a in and b nj. So, first we have end of all of this and then we have a giant OR of all of this and this is the output.

So, clearly this is a constant depth polynomial size circuit. So, it is a constant depth polynomial size circuit and this correctly computes the i jth entry of the product. So, hence a Boolean matrix multiplication is in AC 0.

(Refer Slide Time: 21:20)



The next thing I want to talk about so since it is an AC 0, it also follows that this is in this also in NC 1, because AC 0 is contained in NC 1. The next thing is graph reachability. So, I think we also called it path when we were discussing space complexity. This was kind of the canonical language that captured any that was kind of complete language for any space bounded complexity class. So, we had a directed graph G.

And we had two vertices s and t is there a path from s to t or t reachable from s, this was language. So, we will see that this is in AC 1. So, we will see that this is an AC 1, let us see how. The key point is that we could use what we just saw Boolean matrix multiplication. So, for let us first define the something like the adjacent matrix. So, we will define call it A. so, ij th entry of A is one for two cases.

One is that if i is equal to j that means all the diagonal entries are 1 and two is if there is an edge from i to j. So, there is an edge from i to j. So, in these two cases ijth entry is 1. Another way to interpret A is that can you reach j from i in one or less number of steps. So, one step means there is an edge from i to j less means how can you go from i to j in zero steps if the other possibilities i and j are the same place.

So, ij th entry of A is one if you can go from i to j in one or zero steps one or less number of steps otherwise it is zero. So, the claim is that this is something that can be verified. There is a

path from i to j of length at most l if and only if you take the lth power of a and look at its ijth entry if that is one. So, there is a path of length at most l from i to j in the graph G if and only if the if you look at the lth power of a and look at its ijth then it is one.

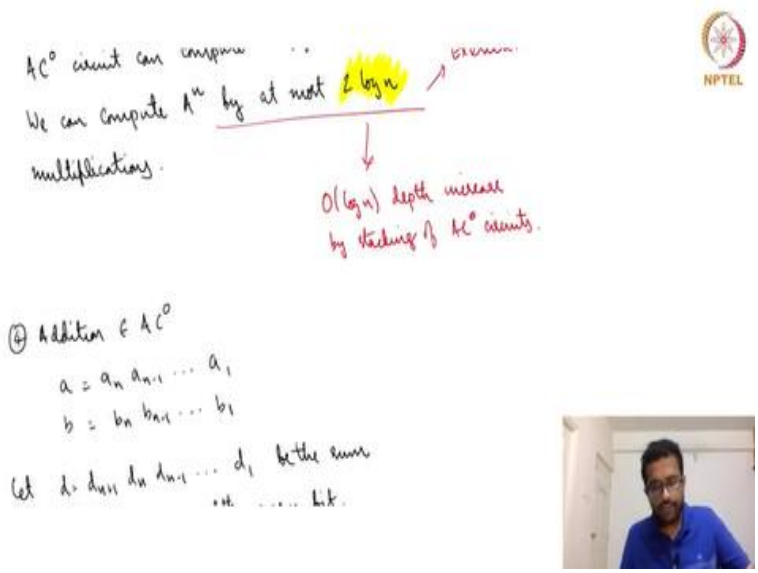
So, this can easily be proved using induction. So, I will just give you a high level idea. So, by definition the claim is true for l equal to 1, the way we have defined a it is true for l equal to 1. A is A power 1 and the way we have chosen the entries for A it is satisfying this requirement. Now let us see what happens when you take A squared. So, what happens when you take a squared? So, when I say lth power or a squared, I am talking about Boolean matrix multiplication there.

So, A square is basically OR k equal to 1 to n a i k and b kj this is what the ijth entry of the A square. So, it is not b kj because it is not a and b it is a kj. So, this will be one if there is a ij equal to 1 if and only if there is a k such that a ik and a kj are both 1. So, which means a we can reach from i to k in one step and k to j in one step. So, which means you can reach i to j in two steps. So, notice that in the case that is exactly what we wanted.

A square, the ijth entry of A square is one if when you leave there is a path of at most two steps from i to j. So, this is what we wanted to verify. Again, this if you elaborate it and use an inductive proposition you can prove this claim. I am not writing the details but you get the idea. So, now we will say that G, s, t is reachable or G, s, t is in path if and only if there is a path of length at most n from s to t. So, the question is t reachable from s.

So, the length of the path is actually at most n in fact it is actually at most n - 1, so a power n - 1 s t equal to 1.

(Refer Slide Time: 26:37)



So, now all that remains is to compute the n - 1 entry, n - 1 power of A and then look at the s, t entry. So, we have seen how to compute is a product of two matrices using an AC 0 circuit. Now we could use repeat the same idea, you could compute A square using that AC circuit and then A power 4 using a squared as input and so on. And if you use the; ideas of this repeated exponentiation repeated squaring for the exponentiation.

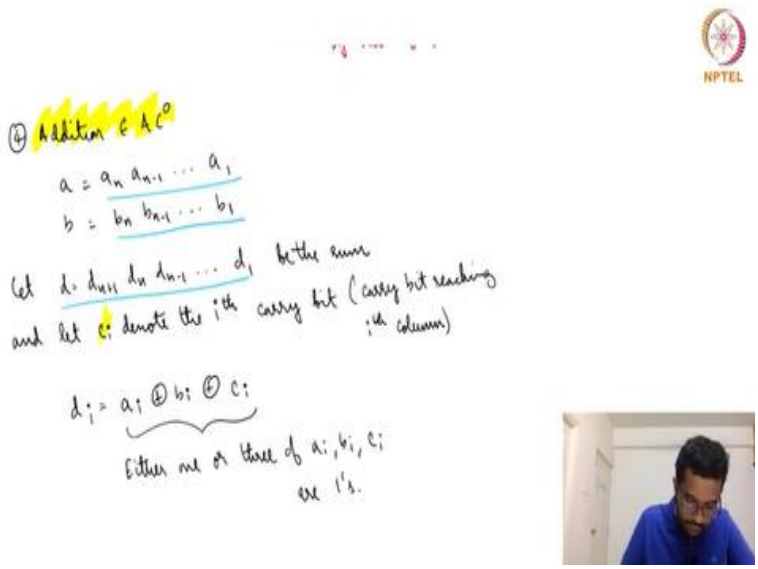
We could compute A power n by using at most 2 log n multiplications. So, you can again this is something not very difficult to see, this is something that you can work out. So, again you can work this out. How can you compute a power n using at most 2 log n multiplication? So, you compute a A square A power 4, A power 8 and so on and then you do something. So, it turns out that is all that you need. So, you need to compute A power n and then look at the s, tth entry.

So, you will use an AC 0 circuit 2 log n times and perhaps the output of 1 is used in turn as an input of the other. So, AC 0 circuit multiplied by 2 log n means the depth can become from the constant size circuit it maybe it can become order log n size circuit or order log n depth. So, this could result in order log n depth increase by stacking of AC 0 circuits and that gives us an AC 1 circuit and that is what we stated at the beginning.

And in fact, I would also say that this means that it is in NC 2 because AC 1 is contained in NC 2. And in fact, it is I am not saying it explicitly but it is also true that all these examples that we

are doing. All of these are actually uniform NC or AC, because there is a very clear structure of these circuits. And this can be generated using a log space algorithm.

(Refer Slide Time: 29:15)



And finally, I just want to take up one more language or one more operation which is addition and we will show that addition of 2 n bit numbers is an AC 0. So, why is that the case? So let a and b be the 2 n bit numbers, a b a and n - 1 up to a 1 and b n and b n - 1 up to b 1. Let the sum be denoted by d, d and the sum could have one more bit it cannot have more than one extra bit. So, let it be d n + 1 d n dot dot d 1.

So, when we when we add 2 bits or add to binary numbers or add to numbers in general, we could have there is sum and we also could have the carry bit. So, when you add a 1 and b 1, it may generate d 1 and it could also generate a carry bit that goes to the next column. So, let c i be the carry bit that enters into the ith column sorry carry bit reaching the ith place not generated from the ith column but reaching the ith column.

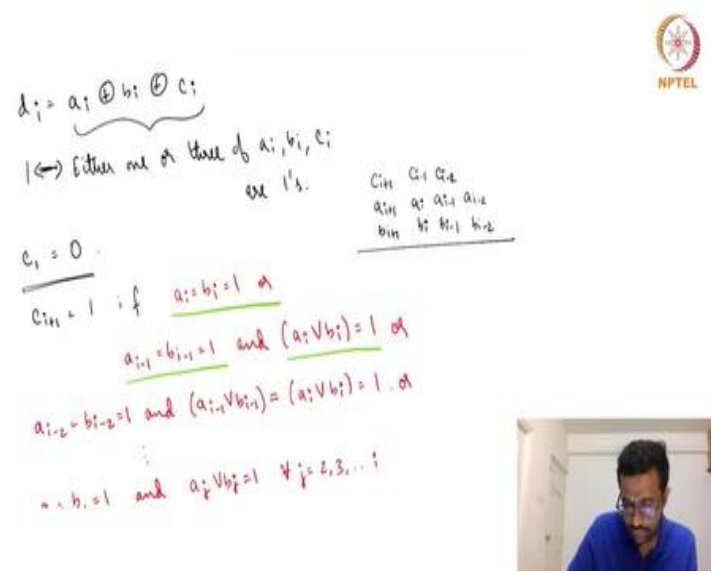
We will just say column instead of place. So, let us we call it c i. So, now let us what is easy is to look what is easy is to look at the sum. So, the sum at the ith bit is basically you have the sum at the ith bit is you have a i, you have b i and you have a carry coming up from the next from the previous location and you need to output the sum. And in fact, the sum is just a parity function on these three quantities on a i, b i, and c i.

So, the sum is merely a parity of these three quantities. So, d i is 1 if and only if at least one of a i, b i, c i is 1 or all of a i, b i, c i is 1. If 0 or 2 of them are one then d i is 0 if 1 or 3 of them are 1 then b i is 1. So, I will say d i is one if and only if either 1 or 3 of a i, b i, c i's are ones. So, it is a parity function. So, again parity function we have already seen how to write that using a constant size circuit. So, again this is a constant size circuit so this is straight forward.

What one may wonder or one may think is the difficult part is the carry. Because let us look at the carry at the nth bit the carry c n. So, the carry at the nth bit arises as a result of the carry from the n – 1th bit addition which arises as a result of the carry from the n – 2th bit addition and so on. So, one may be tempted to think that there is an n - 1 or n level dependency and a n and the c n depends on a n - 1 and b n - 1 which in turn depends on n - 2 and b n - 2 and so on.

So, you one may think that there is a dependency that goes down to n levels. So, one may be tempted to think that the depth of s resulting circuit may be actually order n or linear in size or linear in depth. So, this will mean that it cannot fit in the NC hierarchy or AC hierarchy. Because all our NC or AC hierarchy we have the depth equal to order log n power i. So, this cannot be linear. However, it turns out that we do not need an dependency order n dependency.

(Refer Slide Time: 33:41)

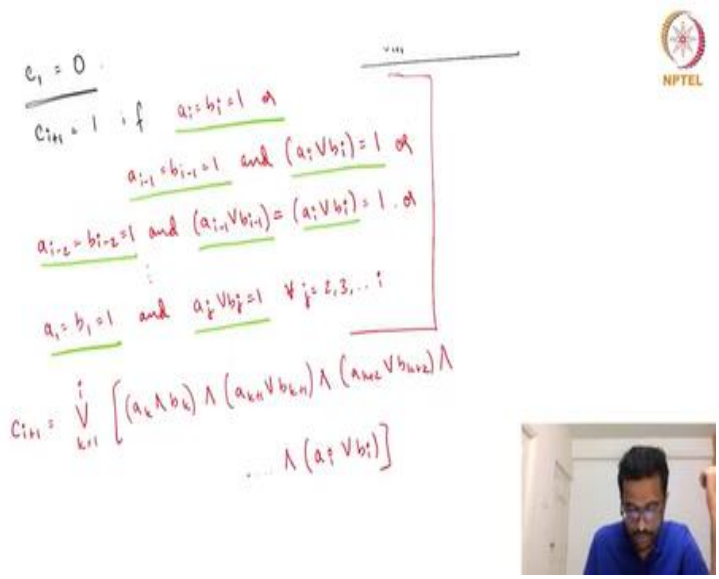


Let us say y. So, when is the ith carry bit equal to 1? So, let us see. So, c 1 is 0 by definition because a 1 and b 1 are the least significant bits. So, there is nothing coming from the from the 0th carry. So, c 1 is 0 and c i + 1 is 1 when you c i + 1 equal to 1, it becomes 1 c i + 1 is I will maybe write down here c i + 1 is the carry at the a i + 1 and b i + 1. So, let us look at what would have happened at a i or b i that resulted in c i + 1 being 1.

It is possible that a i and b i are both ones, if a and b are both ones certainly c i + 1 = 1. Other possibility is a i and b i are not both ones but at least one of them is a one. Let us say one of them is a one and but then they got a carry from c i - 1 which means they got a c i - 1 as a carry. And how could c i - 1 have come? That could have come because perhaps a i - 1 and b i - 1 are both ones.

So, I have captured these two things over here, c i + 1 = 1 if both a i and b i are ones or one of a i and b i are 1 is 1 and both a i - 1 and b i - 1 are ones. Now you could take it to the next level. So, another possibility is that both at least 1 of a i - 1 and b i - 1 are ones but they got a carry c i - 2. And how did they get a carry? Because both a i - 2 and b i - 2 are ones.

(Refer Slide Time: 36:00)



So, you could write that as a next step, sorry a i - 2 and b i - 2 are ones and at least one of a i - 1 and b i - 1 are 1 and at least one of a i and b i are ones. And notice that these conditions are also important suppose a i - 2 and b i - 2 are both ones you get a carry c i - 1, but both a i - 1 and b i -

1 are both 0s so then that carry does not propagate. So, for the carry to propagate at least one of a i - 1 and b i - 1 has to be 1 and one of a i and b i has to be 1.

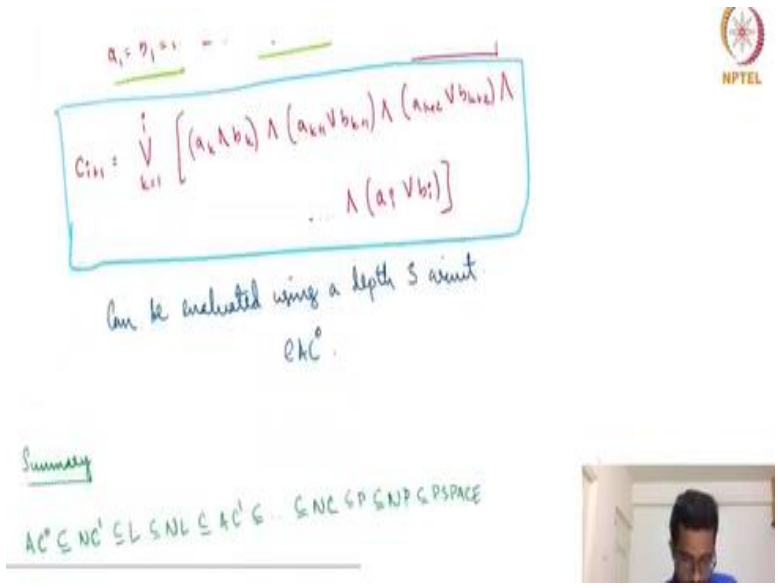
And now you can keep generalizing this or we can keep extending this and the last possible situation is that a i and b i are both ones sorry a1 and b 1 are both ones and for the rest at least one of them are one so the carry keeps propagating. So, a 1 is 1 and b 1 is 1 and then for a 2 and b 2 at least one of them is one a 3 and b 3 at least one of them is one and so on till a i and b i for all j both a j and b j at least one of them are 1 for all j equal to 1, 2, 3, 4, 5 up to i.

So, this is how the carries can occur, this is only situation. So, now let us write an equation for this a Boolean expression for this. So, c i + 1 is equal to a big OR. So, OR is basically we are trying to whatever we have written over here this big red expression we are trying to write it more crisply over here. So, it is an OR of all these things. So, what is the so maybe it will help look at the k equal to one case.

So, when k = 1 we have a 1 and b 1, both of them should be one which is actually the last row over here, a 1 and b 1 and I think there is a small mistake here which actually be k + 1, I think. So, a 1 and b 1 are both 1 and then a 2 and b 2 at least one of them has to be 1 and then followed by a 3 and b 3 at least one of them has to be 1. So, maybe I will just write one more a k + 2 or b k + 2 and so on till a i or b i.

This is when k = 1 and k = 2 it starts from a 2 and b 2 should both be 1 and then everything from that point till I should be at least one of them should be one.

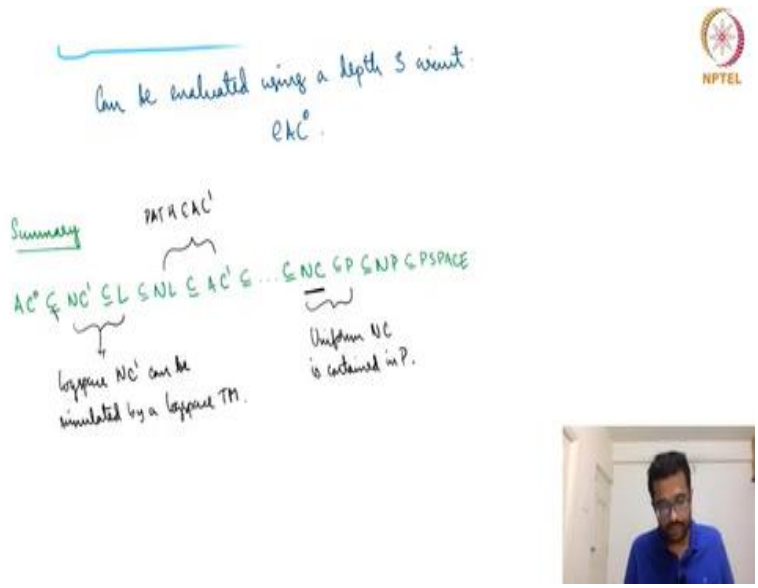
(Refer Slide Time: 38:58)



Any way you can verify that; this expression for c i is the correct expression this expression that I have enclosed. And now it is very straightforward to see that this expression can be evaluated using a constant depth circuit. So, you have a big OR but that is we are allowed unbounded fan in and in fact there is a big and inside each of these ORs that is also because its unbounded fan in. And inside this nth it is all small circuits it is all fan in two items.

So, we will get a depth three circuit for this. So, this can be evaluated using a depth three circuit. So, constant depth circuit and send outs. So, it is in AC 0 this is what we had claimed. So, addition is in AC 0 Boolean matrix multiplication is an AC 0, graph reachability is in AC 1 and parity is in NC 1.

(Refer Slide Time: 40:19)



And I would like to just summarize what we know so far or some of which I will just give a brief outline about these classes and how they relate to some of the classes that we already know. So, one point is AC 0 is contained in NC 1 that NC 0 is not interesting. So, I will not write about that and this containment is strict AC 0 contained in NC 1 and NC 1 is actually contained in log space. This is because so maybe I will just write in.

This is because, again this is only for this is when I am assuming that all these classes where applicable is log space uniform. So, log space NC 1 can be simulated by a log space TM. This is because the generation part can be taken by a log space Turing machine because the generation is in log space and then the circuit evaluation once you generate it even that can be taken care by a lock space Turing machine.

And combining these two the generation as the evaluation takes some work. But we saw the idea at the beginning of space complexity. How you could combine a space bounded reduction? You had this idea about running the like how you could combine two log space bounded machine to get a house space bounded, two space trace bounded machines can be combined to get a space bounded machine.

You will start the second machine and whenever you need an input from the output of the first machine you will then run the first output machine and so on. So, that is the idea over here. L is

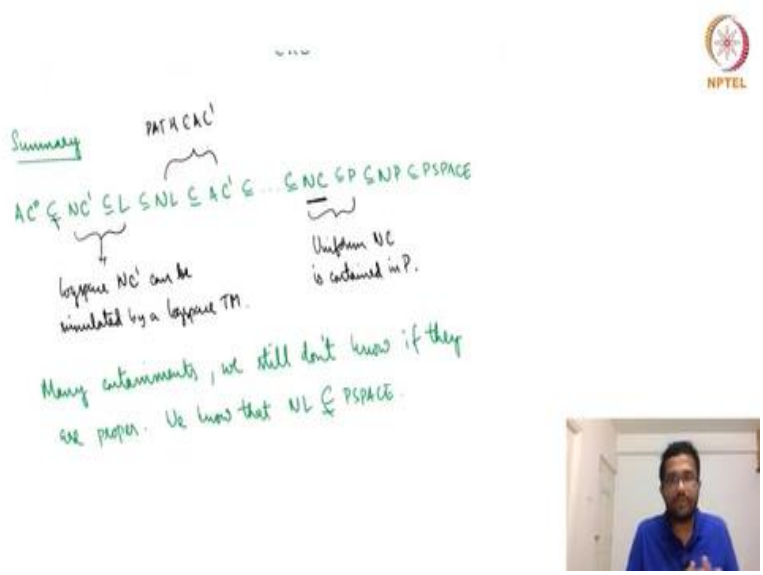
contained in NL which something we know NL is contained in AC 1; this is because of something that we have seen already graph reachability is in AC 1. And we saw that graph reachability is kind of the canonical standard complete problem of NL.

So, this is because path is in AC 1 path or reachability. And then then it is just the NC hierarchy AC 1 is contained in NC 2 and so on. So, then I am writing everything is contained in NC and the uniform NC is contained in P. So, this is because uniform NC is contained in P. So, in the previous lecture I said that any log space uniform family is contained in P or all the log space uniform family is the same as P.

But then in the case of NC you have more restrictions, you cannot guess any circuit of using a log space or you cannot generate any circuit using a log space machine. Because we have other constraints about how much that the depth should be log n power something. So, it is only containment, it is not inequality and the rest are something that we already know P is contained in NP which is both content and P space.

So, this is some collection of some classes that we already know how they relate to each other. Many of these we do not know still which containment is strict or which containment is proper.

(Refer Slide Time: 44:27)



We know that NL is so many containments we still do not know if they are proper. So, we know that NL is a strict subset of P space and I think that is about it. NL is a strict subset of P space this is P space hierarchy. And so that means that some at least somewhere in this chain there should be two classes that are nearby but they are not the same. It cannot be that everything is equal if everything is equal then NL will be equal to P space which we know is certainly not true.

But we do not know which one. I am just trying to see if there is any other thing that we know, any other strict containment that we know seems like no. I think everything else is still open and I think that is all that I wanted to say in this lecture 30. So, we defined the NC hierarchy, NC is the class of all languages that can be decided by polynomial size or order log n to the power i depth circuits using AND, OR, NOT gates and AND and OR gates are have fan in 2.

AC i is exactly the same but we allow unbounded fan in, then we saw this hierarchy between in NC and AC and we saw what is uniform in NC and AC and from then on, we only focus on uniform in NC and AC. So, we saw that parity is in NC 1 and I mentioned that parity is not in AC 0. So, that is why AC 0 is a strict subset of NC 1. We saw that Boolean matrix multiplication is in AC 0 graph reachability is in AC 1.

And we also saw that addition is in AC 0 and then finally we summarized about some of the classes that we have already seen. And just to this completes week 7 just to summarize week 7 we saw definitions of circuits. We saw circuits that require certain size or we saw that most functions can be computed by exponential size circuits and in fact most functions require exponential size circuits. And then we saw the class P by poly and what that is capable of is how P is contained in P by poly.

But then they are not quite the same then we saw Karp-Lipton theorem and which says how when P is related to P by poly. It is unlikely that NP is contained in P by poly. We saw Mayer's theorem about exponential time how that relates to P by poly and then Kannan's theorem which said that there are languages or functions that require arbitrary large size circuits or arbitrary large size polynomials in the polynomial hierarchy and then we saw Turing machines that take advice and finally we saw the NC and AC hierarchy and that completes our week 7. Thank you.