

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

Lecture -17
NL co-NL_ Part 1

(Refer Slide Time: 00:15)

Lecture 17 - NL = co-NL
29 September 2020 10:14

NPTEL

$NL = co-NL$

Result by Neil Immerman & Robert Szelepcsényi
(1987)

* Bit of a surprise. Since we don't have anything similar in time bounded complexity.

We have seen PATH is NL-complete.

$PATH = \{ \langle G, s, t \rangle \mid G \text{ has a directed } s-t \text{ path} \}$.

Hello and welcome to lecture 17 of the course computational complexity. In this lecture we will see an important result in the space complexity theory called which is a result by Neil Immerman and Robert Szelepcsinyi when they proved that NL is equal to co-NL this was proved in 1987 and these two people independently proved these the same result. So, in these times much like Cook Leven independently proved from the US or Canada and the other person in Russia.

Similarly Neil Immweman was in the western side of things and Robert Szelepcsinyi was in the eastern side of things and they independently proved the result NL equal to co-NL this is just like Savitch's theorem where we do not have an analogous result in the time complexity classes this is also a result where we do not have an analogous result. Because an analogous result would have been NP equal to co-NP but unfortunately we do not have that result.

And so, it was believed to be the opposite that NL and co-NL are distinct classes but then it was a bit of a surprise when somebody came along and showed that these two classes were the same. So, let us see how the proof goes.

(Refer Slide Time: 01:42)

$PATH = \{G, s, t \mid \dots\}$
 We will see that $PATH \in NL$. Since $PATH$ is
 NL-complete, $PATH$ is co-NL complete. If
 $PATH \in NL$, this implies $co-NL \subseteq NL$.
 $PATH \in NL \Rightarrow PATH \in co-NL \Rightarrow NL \subseteq co-NL$.
 Thus we get $NL = co-NL$.
 What we need to show is $PATH \in NL$.

So, we have already seen in the previous lectures that path is NL complete. So, what is path? Path is the problem of given graphs G and designated vertices s and t is there a path from s to t in the graph G that is a problem path. Now what we will show is that path is in NL; so, we have already seen that path is NL complete and then we will show that path complement is NL complete. So, the whole proof is going to show that path complement is in NL.

So, let me at the beginning itself let us say let us see why this showing that path complement is in NL is sufficient to show the result that NL is equal to co-NL. So, this is because since path is an NL complete language, since path is NL complete the complement of path is co-NL complete. So, just like we had SAT complement being co-NP complete path complement is co-NL complete. Now if we show that path complement is in NL which is what we are trying to show we are actually showing that a co-NL complete language is in NL which means by reduction anything that is in co-NL can be accomplished or can be reduced to n.

So, this implies that anything that is in co-NL can be reduced to NL or rather co-NL is contained in NL. So, now we want to show NL is equal to co-NL. So, we have one direction of the proof

and for the other direction if path complement is in NL it also implies that path is in co-NL. Because anything that is in NL its complement is in co-NL. So, the complement of path complement is simply path and path is an NL complete language.

So, if that is in co-NL anything in NL can be reduced to co-NL because it is an NL complete language. So, anything can reduce anything in NL reduces to path and that is in turn shown to be in co-NL. So this shows that NL is contained in co-NL. So, we first showed that path in NL implies co-NL is in NL and then showed that path is in an or sorry path complement is an implies that co-NL is in NL is contained NL and then we show that path complement is an implies that NL is in co-NL.

So, together these two implications or these two containments show that NL is equal to co-NL. So all that we have to do is to show, now show that path complement is in NL this is what we will show for the rest of this lecture because this is enough as we have already seen to show that NL is equal to co-NL.

(Refer Slide Time: 05:01)

Thus we get $NL = co-NL$

What we need to show is $\overline{PATH} \in NL$

$\overline{PATH} = \{ \langle G, s, t \rangle \mid G \text{ has no directed } s-t \text{ path} \}$

Has do you get a certificate for these not being a path?

Simple: Given G , the number of reachable vertices

So, what is path complement let us just write it down path complement. So, path is G, s, t such that such that G has a directed path from s to t . So, path complement is G, s, t such that G has no directed s, t path. So, there is no path from s to t in G . So, that is a complement. So, just one small point about complements maybe it is a good time to make that point. So, whenever I say

the complement of a language we are just taking the opposite but there are a lot of inputs that that are not even conforming to this particular specification and we usually do not refer to those inputs at all.

So, it is enough to restrict because these are usually easy to see that they are not in the correct format. So, this can be usually very easily decided. So, when I say path complement I am only talking about inputs in the correct form G, s, t and since path contains all the G, s, t such that there is an s, t path in G path complement contains all the G, s, t such that there is no s, t path in G . So, now to show that path complement is in NL we want an NL machine to show that there is no path.

So, in a way you want an NL verifiable certificate just like we had certificates in NP. Now we want an NL verifiable certificate that shows that there is no path from s to t . How can somebody produce a certificate that there is no path from this vertex to this vertex. If I want to say that there is a path from this vertex to that vertex then I can tell you the path and you can verify that this is a correct path. So, that is okay.

However how can somebody tell you something that you can use to verify that there is no path. So, that is the challenge here. So, that requires some some intelligences some cleverness some smartness. So, first we will actually in this particular part of this lecture we will answer a slightly different question. The question being given c where c is the count of the number of reachable vertices from s .

So, let us say from s you can reach 10 vertices. So, c is a number of vertices reachable from s . Now can we show that t is not reachable from s . So, again it is the same problem and exactly the same problem except that we are giving an extra bit of information which is the count of reachable vertices from s . Suppose somebody is giving us that. So, let me ask you a question suppose the count of reachable vertices from s is 1.

Now is t reasonable? Notice that s is always reachable from itself. So, if the number of reachable vertices from s is just 1 that means the only vertex reachable from s is itself. So, t is not

reachable. So, suppose we know s has a neighbour let us say u and I say that the count of reachable vertices is 2. So, then now we know that the reachable vertices are s and u . So, these two account for the count given. So, that means no other vertex is reachable.

So this is the idea that we will use. So, what we will do is to if a count of 10 is given we will show 10 vertices that are reachable from s and actually verify that they are all reachable from s . So, we are converting the non-reachability into a reachability problem. So, we want to show t is not reachable. So, instead we will show 10 vertices that are reachable. So, and because we know that there are exactly 10 vertices reachable from s these must be the 10 vertices. So, t cannot be one of these 10 vertices. So, t is not reachable.

So, this is how we convert the non reachability into a reachability problem or non reachability verification to reach a verification of reachability. So, this is the high level idea. Now let us see the details. So, given G , s , t and c where c is the count of the vertices reachable from s we need to verify that t is not reachable.

(Refer Slide Time: 09:50)

v_i) sum s

But c could be $O(n)$. The NL machine cannot store all of c vertices.

- Counter that iterates through all vertices.
- At each vertex, NL machine guesses if it is a reachable vertex. If it's a reachable

So, perhaps the NL machine what it can do is to. So, now since we have non-determinism the cases that I explained had was easier cases. So, now we want to guess c vertices that are reachable from s . So, now we have non-determinism what we can do is that we can guess the c

vertices that are reachable from s and verify that they are indeed reachable from s . So, the NL machine can non-deterministically guess c vertices and verify.

So, the non-determinism is used to guess and then verification can be done easily verify that they are indeed reachable from s and one small point when we guess c vertices. So, this none of these should be t , c vertices none of which should be t because if we try to show that if t is one of these vertices that means the t is reachable. So, then that means that this is opposite of what we want to show.

So, all the guest vertices none of it should be none of it should be t . So in other words what we are trying to show is that to show that t is not reachable which is what I have written here in the side we will guess we will show the existence of c vertices none of which is t none of which are t and that all these vertices are reachable. So, to show that t is not reachable we will show c versus other reach.

Now again there are there are implementation details here all this we want to do in non-deterministic log space we want to show path complement is in NL. So, we cannot simply guess 10 vertices or something c vertices because c could be like maybe there could be n by 2 vertices reachable from s and just to write down the n by 2 vertices will require order in space at least order in space. So, we cannot have the NL machine just guess all the n by 2 vertices note them down somewhere and then and then go about verifying them this has to be done in a way. So, that no the space is not the space usage is is carefully maintained. So, how do we do this?

(Refer Slide Time: 12:37)

$f_{min} s$ $f_{max} s$
 But c could be $O(n)$. The NT machine cannot store all of c vertices.

- Counter that iterates through all vertices.
- At each vertex, NT machine guesses if it is a reachable vertex. If it's a reachable

So, what we will do is we will guess a sequence of vertices and ascend when each one of them is guessed to be in a guessed to be a vertex that is reachable from s we will also complete the verification part. So, all that we will do is to keep track of the counter and we will go about these it is choosing these vertices in an order, so, that we will not end up guessing a vertex two times. So, we go. So, each vertex is either picked or not picked once it is picked we will verify that there is a path.

If there is if this cannot be verified then this whole computation thing collapses because it is non-deterministic we want to show that path complement is in NL which is not deterministic then all we need to do all we need to do is to show some way of guessing and correctly verifying. Some sequence of guesses lead to a correct verification this is all that we need to do. So, what we have is a counter that iterates through all the vertices.

A counter that iterates through all the vertices at each vertex, so, the counter could be let us say 1 to n if there are n vertices. So, each vertex is a number from 1 to n . So, when you are at each vertex the machine can guess whether it is part of a reachable vertex from s or not. So, let us say the first vertex is guessed to be reachable. If it is guessed to be reachable we also guess a path from s to that vertex s to vertex 1. And once we get a path we can verify that this path is a correct path.

If vertex a vertex say vertex number 1 is guessed and correctly verified we increment a count. If we guess if anywhere our guesses go wrong. So, even in guessing whether it is reachable or in verifying then that whole computation path is rejected if there is any mismatch anywhere that whole computation path is rejected. All that we need is some computation path that leads to correct verification it there could be 100s and 1000s of other paths other computations that could lead to reject that is okay.

All we need is one correct path there is a path I mean computation path. So, at each instance we guess whether a vertex is reachable let us say vertex 10 is reachable we correctly guess a computation correctly guess a path from s to that vertex 10 and verify it and then the counter is incremented. If we guess if we cannot correctly verify it or so, that can happen due to reasons either the guest path is incorrect or the vertex is also if the vertex is reachable but the path is incorrect or the vertex itself is unreachable.

Whatever be the case we the entire path entire computation is rejected is terminated. But if there is there will be a correct set of since we know c the number of guesses number of words that are reachable there will be a sequence of guesses that will lead to correctly identifying these c vertices.


(Refer Slide Time: 15:53)


is one set of guesses that will help verify this.

Algorithm 1 NL algorithm to which accepts if t is not reachable from s , given c .

1. Let $d = 0$.
2. for each vertex $u \in G$ do
3. Nondeterministically either perform or skip the following steps:
4. Call the function CHECKPATH($G, s, u, \{t\}$).
5. If $u = t$, then reject.
6. $d \leftarrow d + 1$
7. If $d = c$ then
8. Accept.
9. else
10. Reject.

Space usage: Need to store d and u .
 $\rightarrow O(\log n)$





So, and since t is not reachable if t is not reachable we will correctly identify this c vertices and none of it will be t and we will we will accept because we need to show that t is not reachable. So, if t is not reachable that needs to accept this is what we will do this is a high level idea. So, let me just say it once again. So, and there is this part. So, this is a small bit. So, d is initial d is the count of the vertices reachable from s .

If we assign it to zero. So, I am talking about item number one here we initialize to zero and for each vertex u in the graph we either select it to be reachable or we select it to be an unreachable vertex if it is selected to be a reachable vertex then we call the function check path of G, s, u, V check path what it does I will just briefly explain now and later we see it in detail. It will check whether there is a path from s to u that of length at most V size of V this is what it does.

So, if it is a vertex that is reachable then we will guess it correctly and we will call the function and if at any point the guessed vertex if u is guessed to be reachable sorry t is guessed to be reachable then the whole computation path is rejected because t is something that we do not want to reach. If at any point t is found to be reachable we do not proceed with that computation and for every vertex that is correctly verified to be reachable we increment d .

At the end we see if we have not rejected the computation then we see what is the count? Is the count actually equal to the actual count that is given to us which is c . So, recall we are given G, s, t and the actual count of vertices reachable from s which is c . So, we check whether we guessed c vertices that are reachable from s . If we guess the c vertices that are reachable from s none of it being t we accept otherwise we reject.

So, if t is not reachable this will accept exactly when t is not reachable and we are able to guess the vertices that are not the that are the vertices reachable from s . Because if t is reachable then the count of words is reachable from s will include t and the algorithm will break because we also check that none of the words is reachable r equal to t okay. So, what is the space usage here?

(Refer Slide Time: 18:57)

Space usage: Need to store d and u .
 $\rightarrow O(\log n)$

List of 8 vertices reachable from s

For each such vertex v , the path from s to v

$c=8$

t

NPTEL

Let us see what is required to be saved here d is a counter that needs to be saved and u is a counter u is a vertex you need to check for each vertex u equal 1 to 3. So, that is also a counter. So, d and u are the things to be that occupy space. We also call the function here check path. So, the check path will use some space but that we will worry about when we actually see the function check path.

So, right now we do not let us not worry about that because we have already seen that calling a function if a program if a machine runs in NL space or logarithmic space and you can always make a function called the function call may also be NL space and you can always do that this we have already seen at the beginning when we showed the why the log space reductions make sense. So, you can combine these but anyway the check path function we will see very soon.

So, the only things that we need to save here are u and d , d is a counter of length at most c . So, of c which is at most c and c is a number less than the number of vertices in the graph which is less than n , u is also a counter that will go up to at most n . So, both of them are numbers at most n so, can be represented in $O \log n$ bits. So, this total space usage is $O \log n$. So, just to illustrate, so, suppose this is s and c is 8.

So, which means there are 8 vertices reachable from s including itself. So, so these 8 vertices are guessed and we correctly get guess these 8 vertices and verify the paths. So, now this means the t

is not reachable because these are the exactly the set of vertices that are vertices that are reachable from s . So, what we are doing here is to like first we guess the list of 8 vertices reachable from s and this is not done in a collective manner.

We not guess all the words is reasonable we just do that one by one and then for each such vertex v that is guessed to be reachable let we guess the path from s to b and this is not done in the code that we showed right here because that is being done by the function check path. So, here what we do is we one by one we guess each vertex to be reachable or not reachable if it is reachable we guess the path.

And if any of these does not work out either we guess an unreachable vertex to be reachable or a reachable vertex to be unreachable or t to be reachable or we guess that everything correctly but one of the paths from s to a vertex p is incorrectly guessed anything of this happens that computation path is cancelled. All that we need is some computation path gets all the cases correct and that is enough for us.

So, then we can verify that t is indeed not reachable from s by guessing the correct set of vertices that are reachable from s . Now what remains is to see the function check path how do we check that a vertex there is a path from s to u in V steps.

(Refer Slide Time: 22:59)

NPTEL

Algorithm 2 CHECKPATH(G, s, u, k)

- 1: $w' = s$.
- 2: for $j = 1$ to k do
- 3: Nondeterministically guess a w , and reject if $\{ \{ (w', w) \notin E(G) \} \text{ AND } (w' \neq w) \}$.
- 4: if $w = u$ then
- 5: Accept and return.
- 6: $w' \leftarrow w$.
- 7: $j \leftarrow j + 1$.
- 8: Reject.

Space usage: Need to store w, w', k ,
All need $O(\log n)$.

So, check path checks if there is a path from s to u in the graph G with at most v edges. So, again I have already kind of explained what it does but. So, this is a very simple subroutine there is not much high level ideas here all we do is something like this. So, suppose s is this and u is this we just guess a sequence of vertices let us say $w_1 w_2$ and something w_1, w_{10}, w_6 some something like that and we keep moving ahead and we check if we reach u at any point. So, what we do is we guess a sequence of vertices.

Let us say w_1, w_{10} something. So, for each successive pair we check whether it forms an edge. So, what happens is initially we call s to be equal to w_1 and sorry w_{prime} and we guess a w . So, let us say the guess w is w_2 and then check whether there is an edge from w_{prime} to w . So, now in this case s is w_{prime} , w_1 is w . So, there is an edge, so, fine. Now what we do is we rename w to be w_{prime} not just that we also increment a counter.

So, now we have verified one edge we increment the counter to one. So, the counter here is the counter that is used here is j . So, j is equal to 1. Now . Now this is w_{prime} we guess another w . So, let us say this w_{10} is decide guessed check whether there is an edge from w_{prime} to w . So, w_1 to w_{10} indeed there is an edge. So, j is incremented to 2 and w is again renamed to be w_{prime} .

Now let us say we guessed w to be let us say some other vertex let us say some w_{15} from which to which there is no edge. In this case we will check that there is an edge and we will not see an edge. So, this computation path gets rejected this entire computation goes. But the point is if there is a path from s to u we will there will be a sequence of guesses that will help us verify that s to u is reachable from s .

Finally why do we have the counter here that is because we know that the we are checking whether the path is of length at most k check path is called for G, s, u, k whether there is a path from s to u in G of length at most k . So, here we use at most here we use V but we will we will see that we will need this for other variables that is why it is given as an argument. So, we are trying to see whether there is a path of length at most k .

In fact there is a small typo here no there is no type ok sorry we not eternally secretly non-deterministically guess the next vertex and keep moving ahead. If there is a path of length at most k we will find it.

(Refer Slide Time: 26:46)

Now we have seen how to check using an NL machine that t is not reachable, provided the limit of reachable vertices are given.

Q: Given G, s , how to determine c , the no of reachable vertices?

We say an NL machine computes a function f correctly if it either

So, if the counter, so, this is this part is written for size V , k is equal to size V . So, maybe I will just modify it slightly. So, what is check sorry what is check path G, s, u, k and rejective counter becomes greater than k . So, when counter becomes greater than k means there is we are exceeding the limit. So, we will stop. So, this is all that we do, if we keep progressing this pointers w prime ahead with incrementing the counter and it we accept if the counter j does not cross k and w prime reaches the target destination u .

So, if at any point w prime or w is equal to u then we accept otherwise we reject. So, there are two things checked here whenever w is guessed we reject if there is no edge from w to w prime and if w is not equal to w prime. If w is equal to w prime then it is fine that that leads to an accept. So, basically this is the algorithm to check whether there is a path from s to u of length at most k .

So, you guess a w we reject if w prime and w do not form an edge and w prime is not equal to w . So, in other words this is the using this de Morgan's laws it is nothing but w prime w is is an edge or w is w prime this whole thing the negation of this whole thing is what we have. So, we

are if either of this is the case we reject if the opposite of this happens the complement. And we accept if we reach w equal to u otherwise we reject when the count of the the steps crosses k .

So, what are the things that we need to keep track here the things that we need to keep track is w prime w which are variables either of and then j which is a count. The count will be less than at most the number of vertices n and even for w and w prime these are labels of length of which it can be at most n if you use the numbers to name the vertices. Even if you use something else it will be something similar.

And each one of them is at most n and can be can be written in a $\log n$ space register or memory location. So, each of them require $\log n$ space. So, total we need $O(\log n)$ space there are constant number of items that require $\log n$ space again we use non-determinism but are in $\log n$ space. So, we have seen given a count of number of vertices reachable from s how to verify that t is not reachable from s .

By guessing a set of vertices reachable and then verifying that they are indeed reachable and none of these vertices are t we verify that we verify that t is not reachable. So, now the question remains we started with this assumption how do we that the number of words is reachable is given but. Now how do we compute initially the original problem statement did not have this c provided to us.

Now how do we estimate c or how do we come up with c using in non-deterministic log space machine. So, that part I think I will defer to the next lecture video just for ease of keeping the videos short. So, just to summarize here we wanted to verify that t is not reachable from s . So, we saw how to do that. If we are given the count of vertices that are reachable from s we if you are given the count we guess if c is the count and c is 10 we guess 10 words is reachable from s .

None of which is equal to t we just guess them and verify that they are each other. So, this is what we have seen. So, far and in the next lecture video I will explain how to come up with the number of vertices reachable from s in non-deterministic log space.