Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

W12L67_Results in Space Complexity

Hello and welcome to lecture 60 of the course theory of computation. In this week, week 12, we have been seeing various aspects of space complexity. First, we defined space complexity, and we saw the relation and connection between space complexity measures and time complexity measures. Then we saw L, NL, and NL completeness. Then we saw Savitch's theorem in the previous lecture. In this lecture, I will just summarize or I will just quickly go over some of the other topics that appear in this chapter, in this IPSA book because we do not have the time in the rest of the time that I have in this course.

We do not have the time to completely cover these topics. So, what I will do is whatever the remaining main two topics are there, these two main topics, we will just give an overview of these two main topics. So I'll explain what these results are and what they are about. And hopefully this will help you get some perspective so that in case you are interested, you can go and read those topics yourself.

Or refer to other wonderful resources like videos by other people which explain these topics. So the first thing that I want to talk about is this result that states that NL = co-NL. So, this is one of the topics. But before saying NL = co-NL, I should first say what is co-NL. So, you may remember we defined this term called co-Turing recognizable.

So, this was sometime in chapter 3 or chapter 4, I think. Co-Turing recognizable, a language is said to be co-Turing recognizable if its complement is Turing recognizable. So, co stands for complement. It is similar here. So, co-NL, a language is said to be in co-NL if its complement is in NL.

So, that is the definition of co-NL. Similarly, we can define co-NP, the set of languages whose complements are in NP. So, I did not mention this in this course. NP and co-NP, we do not know the relation between them, meaning both of them contain P. So, it is something like this.

So, if this is NP, this is co-NP, we know that P is contained here, the intersection of NP and co-NP, but we do not know any other relation between NP and co-NP. What I want to say is it is not clear what the relation is. So, there are two possibilities: one is that NP and co-NP could be equal, or it could be that P is equal to the intersection of NP and co-NP. But if one of them is shown to be the subset of the other, let us say if NP is shown to be the subset of the other, then it follows that they are equal. It is not that hard to see, but because we have not covered those topics, I will not get into that.

So, this is what we know: in the case of time complexity, NP and co-NP are not the same or are not known to be the same. They may be the same, but it is not known, right? So, in some sense, we can think of NL and co-NL as some kind of parallel to NP and co-NP, right? Because it's like you have a language, you have an order deterministic class and the complement of its, and the complement class, right? It is not the complement of NP. Again, NL is not the complement of, sorry, co-NL is not the complement of NL. Rather, it is a set of languages whose complement is in NL. Similarly, NP, co-NP is not the complement of NP.

It is a class of languages whose complement is in NP. So, in the case of time complexity, we do not know what the relation is. But in the case of space complexity, there was yet another surprise, which was that NL and co-NL are the same. This was proved in '87 by two independent efforts by Neil Immerman and Robert Szelepcsenyi, right. So this was somewhat surprising.

And this was shown by proving that, so we saw that path is an NL-complete language. Path was an NL-complete language. So, what the proof does is to show that the complement of path is in NL. This implies that since the complement of path is in NL, since path is NL-complete, we know that the complement of path is co-NL-complete. Anything in co-NL can reduce to path complement.

So, again, I have not defined what is co-NL-complete, but it is defined exactly like NL-complete, but with a complement. So, the fact that path complement is in NL and path complement is in co-NL-complete implies that anything in co-NL is contained in NL. So, this gives us that co-NL is contained in NL. And the same argument can be flipped because path complement is in NL implies that path is in co-NL and we know path is NL-complete. This gives us that NL is in co-NL.

So, we have co-NL is in NL, contained in NL and NL is contained in co-NL. Together we get that NL is equal to co-NL. So that is the very, very high-level picture of the proof. So we need to actually, so to show that, so think about it. We saw several examples of non-deterministic languages in NP.

You want to show that there is a subset with a certain sum. You can demonstrate the subset, and it can be verified that it has a certain sum or you want to show a graph is 3-colorable, you can demonstrate a coloring and then you can verify that it is indeed a 3-coloring. But here the problem is difficult. So, like we solve paths in NL, where we kind of verify that there is a path.

But here we have to verify that there is no path. So, now you may be appreciating the difficulty. When something, to verify that something is there is easy. You can guess that and verify it. This graph is three-colorable, so you guess the three-coloring and if it happens to be correct, you can verify it.

Now you have to verify something is not there, the non-existence of a path. How do you verify that there is no path from S to T? You cannot guess a non-path. There could be another sequence of vertices which is an actual path. So this is the tricky part. How do you guess something and verify something which actually tantamounts to a proof that there is no path from S to T?

That happens to be the tricky part. And what the proof does is to actually determine the number of vertices that can be reached from S. So suppose the graph has 100 vertices. Suppose the graph has, let us say, 100 vertices and let us say only 65 of them are reachable. So, this 65 reachable from S.

Of course, this will contain S itself. S is kind of trivially reachable from itself. So you first determine this number 65 and then you guess each of the 65 vertices and verify each of the 65 vertices are indeed reachable and also ensure that none of them is T. So we know that there are 65 vertices reachable. We guess 65 vertices, verify each one of them and verify that all of them are reachable from S and also verify that none of them are T.

So now, we know that 65 vertices are reachable. We verified 65 vertices that are not T, they are all reachable from S. So, it has to be the case that T is not reachable. So, this is how we end up providing a proof that T is not reachable. So, this also involves multiple steps.

First, we have to determine the number of reachable vertices. That is also not easy because remember we only have log space. We have non-determinism, but we only have log space. So determining the number of reachable vertices or in this example, the number 65, that itself is not so straightforward. So we have to kind of slowly build that number.

And once we have that number, how do we actually verify that? You cannot have a list of the vertices reachable from S because that list itself will not be logarithmic in size, right? So, that could be linear in size. So, you have to be clever in doing all of this, but this is the high-level picture, right? So, that is the way we prove that NL is equal to co-NL by giving a certificate that a vertex t is not reachable from a vertex s by this, like, circum to s root, right? I would not say circum to s because it is a difficult task. So, any root is going to be hard, right? So, that is, and again, it was a surprising result that NL and co-NL are the same because one of the kind of similar set of classes that we have in time complexity, P and NP—sorry, NP and co-NP—we do not know of such a result.

So, this is one result. The next thing I want to talk about is PSPACE completeness. So, we talked about NL completeness. So, now similarly, we can talk about the hardest problems in PSPACE. PSPACE is the class of all languages that can be decided in deterministic polynomial space, and because of Savitch's theorem, this is also the same as non-

deterministic polynomial space. So, non-deterministic space n^k is contained in deterministic space n^{2k}.

So, if it is polynomial, non-deterministic and polynomial deterministic space are both the same. So, PSPACE—for example, SAT is in deterministic space n; breadth-first search is in deterministic space n the way we do it—and we also know, like I already mentioned, that PSPACE is equal to the non-deterministic polynomial space, right? And one small fact is that we know that L is not contained in P; L is not—not that it is not contained, L is a strict subset or a proper subset of PSPACE, right? L is log space and P is PSPACE. This is a proper subset, meaning we know that there are languages that are in PSPACE but are not in L, meaning there are languages that require more than logarithmic space but less than polynomial space, right.

We know that there are such languages. This is by something called the space hierarchy theorem. So, which again is another topic that I will not cover. Space hierarchy theorem basically says that if you have two functions, let us say $n^3$and n^2. So, n^3 is clearly bigger than n^2 and even meaning limit $\frac{n^3}{n^2}$.

It tends to infinity. So, if you have two such functions, one clearly faster growing than the other—not by a constant, it has to be even bigger than a constant—then there is a language that requires n^3 space but cannot be decided in n^2 space. So basically, whenever there are two functions, one of which is much faster growing than the other, there is a language that requires a bigger space and cannot be decided in the smaller space. So maybe before I forget, I will just write down a listing of languages or the classes that we know. Classes ordering that we know. So, L is the smallest; L is contained in NL, which is equal to co-NL, and NL is contained in P, which is contained in NP, which is also actually contained in co-NP.

But we don't know—like NP between NP and co-NP, there is no such thing, right? It may be equal, but we don't know. P is also contained in co-NP, and both of these are contained in—there are other things, there are some—there are other things which I will skip. But all of this is going to be contained in PSPACE, which is also the non-deterministic PSPACE. This is just like ordering of the classes that we know. And the thing is that all of these inclusions, right, containments, we don't know whether it is proper or not. We don't know if there is a language that is in NL but not in L. We don't know if there is a language in P but not in NL. We don't know if there is a language in NP but not in P.

We don't know P versus NP. We don't know P versus co-NP. So, all of this we do not know. We do not know whether P is—how P and PSPACE are related. We do not know how NP and PSPACE are related. Is there a language in PSPACE but not in NP? We do not know.

The only thing that we know, however, is this. Out of all these classes, there are some things that I have listed to be equal: NL, co-NL, and PSPACE. The only other thing that we know is L is not equal to PSPACE. That's what I mentioned because of the space hierarchy theorem. L is not equal to PSPACE. So, meaning there are languages in PSPACE but not in L.

So, just think about it. You have a sequence of sets: this contained in this, this contained in this, this contained in this, and so on. But we know that the leftmost and the rightmost are different. So, it cannot be that all of these are equal. There has to be some two sets or some two classes here which either L is not equal to NL or NL is not equal to P or P is not equal to NP or NP is not equal to PSPACE. If all of them are equal, then L will be equal to PSPACE.

So, there—but we do not know which one is not equal. Maybe all of them are not equal. But all of them cannot be equal, but all of them can be not equal because if all of them are equal, then that would mean L equal to PSPACE, but all of them can be not equal. That is possible, right? So, this is just to give you a perspective of what all classes we have learned so far and how they kind of relate to each other, right? Anyway, so now let me come back to the PSPACE, the discussion that I was talking about. So, we know that L is in PSPACE but not equal to PSPACE.

Anyway, so PSPACE completeness is similar to NP completeness or NL completeness. So, B is PSPACE complete if B is in PSPACE and all the languages in PSPACE are reducible to B. And in PSPACE completeness, we use polynomial time reduction. So, this is something that you have to keep in mind, polynomial time reduction.

In NP completeness, we use polynomial time reduction. In NL completeness, we use log space reduction. So, everywhere we want the reduction to be a bit weaker than the class. So, in NP completeness, we want it slightly lower. So, instead of NP, we use polynomial time. In NL completeness, instead of NL, we use logarithmic space—not the non-deterministic one, but the deterministic one.

In the polynomial space completeness, we use a bit lower, which is polynomial time. So, this is the definition of PSPACE completeness. So, these are kind of the hardest problems in PSPACE. I'll just briefly explain a few PSPACE complete languages and summarize. So, one of them is TQBF, otherwise known as true fully quantified Boolean formula.

So, it's a Boolean formula with quantifiers. So, for instance, let's say there is a Boolean formula like satisfiability can be written as does there exist an x_1, does there exist an x_2, such that does there exist an x_n, such that $\phi(x_1, x_2, \ldots, x_n)$ is true. So, basically, we are asking whether there is some assignment x_1, x_2 such that a formula evaluates it true. So, this is set, but now what if we allow both quantifiers? So, instead of does there exist, we also include, let us say, for all, the for-all quantifier. And we ask questions like this: for all

x_1, does there exist an x_2 such that there is an x_3 such that for all x_4 and so on and some other formula $\psi(x_1, x_2, \ldots, x_n)$. This is equal to true.

These types of questions where there is a quantifier. So, quantifier means these things. Quantifier means "there exists" and "for all." "There exists" is called the existential quantifier, and "for all" is called the universal quantifier.

So, you want to quantify each of these variables. All the variables have to be quantified. That's why it's called a fully quantified Boolean formula. So, it has to be a true fully quantified Boolean formula.

So, that is one language which is PSPACE complete. The second one is a bit more fun or easy to understand. So, think of a game where I say the name of a place. Let's say I say Hyderabad. Now, you have to name a place that starts with the last letter of the place that I mentioned.

So, I said Hyderabad. So maybe next, you have to start with D. So, you can say Dubai or you can say Delhi. Now you say Delhi, then I have to name a place that starts with an I. Let's say I say India. Then you say A, A for Austria.

So, then again A, A for let's say Afghanistan. Then N, you say Netherlands. So like that it goes. The only rule is that we should not repeat a place that has already been said. And it has to be a valid name of a place.

So, this game in the general sense is PSPACE complete. So, you may be wondering what is this very specific game? How can this be PSPACE complete? So, that is what I said. This is just to illustrate the game. In general, it is like you have a graph. So, you have a graph like what we have here.

And player 1 will pick, like player 1 will start. So, player 1 will take an arrow, and player 2 has to take an arrow from there. And whoever cannot have a next turn loses. So, for example, in this game that is drawn here, player 1 actually loses.

Because player 1, suppose he... Suppose player 1 takes this way, he goes to 3. Now player 2 can, let's say, go to 6. So this is 1, this is 2. Player 2 can go to 6.

And now player 1 cannot go; there is no outgoing arrow. If player 1 had taken this way, he went to 2. Now player 2 can do this; he can go to 4. Right now, player 1 has only two options: either go to 5 or go to 7. So, suppose he goes to 5. Now player 2 goes to 6, and player 1 is stuck. Right now, player 1, if he had taken this way to 7, then player 2 can go to 9, and again player 1 is stuck. So now you see, whichever step player 1 takes, player 2 has an answer by which he will eventually end up cornering player 1. So, in this particular layout, player 1 does not have a winning strategy.

So, this is a simple game that is easy to describe. So, if player 2 can always win or has a strategy to always win, player 1 cannot have a strategy to always win. What I am saying is here there is a strategy for player 2 to always win. So, whatever the first move that player 1 does, either this or this, player 2 has a response by which eventually he will corner player 1. So, if player 1 takes this as the first move, then player 2 wins immediately. If he goes here, then player 2 goes here, and now depending on whichever one he goes, 5 or 7, player 2 has a response.

So, this shows that player 2 has a winning strategy. So, the question is: given a graph and given such arrows, does player 1 have a winning strategy? So with a designated start vertex, so in here this is a designated start vertex. So how is it related to geography? So geography was like naming some places, etc. Because if you give me a dictionary of names of places, I can, let's say, fix a starting word. and then I can, like, because I know the relation, right, and the goal is to not repeat a place. So, like here we cannot come back to the same vertex. So once you have the list of places, so I have the name Hyderabad, so now I know the next place has to be Dubai, Delhi, Denmark, I don't know what other places that name, like Dhaka, that start with D.

So I know the arrows once I know the names of the places. So, a fixed list and the spellings define the graph for me. So, the question is, does player 1 have a winning strategy? And this happens to be a PSPACE complete problem. In fact, many games are PSPACE complete.

So this is maybe right here. So, generalized geography is PSPACE complete. This is also TQBF is also PSPACE complete. So, in fact, there are many games, many general versions of games that are PSPACE complete. For example, you can define a generalization of the game chess. So, chess is played on an 8 by 8 board, 64 squares, and with 16 pieces on each side to begin with, so 32 pieces overall.

It is a finite game. Right? Because there are rules of the game like let's say if you repeat the same position three times then it's a draw or if you make several moves without a capture or a pawn moving then it is a draw. Right? So there are rules that kind of in like bring bring like enforce some progress which means the game will end after a while or after some fixed number of steps. So chess is a finite game. So there is no, you cannot really ask if chess is like if there is a white win or the black win on chess or whatever. It's a very finite game. So it's like you know the game tic-tac-toe, right? Like you have to put knots and crosses and whoever...

... Whoever gets one in a row, they win. But then it's a finite game. You can draw the entire strategy tree very quickly. So that's not very interesting because you can solve it. But the game tree for chess itself is very complex. But what I'm saying is it's very complicated.

Even with the current computers, you cannot solve it. But the point is that it is finite. It is complex, but it is finite. But you can define a generalized version of chess, like played on an n by n board with, let's say, two n by two n pieces or something. And such a variation has been shown to be PSPACE complete. So a generalized chess, which is general chess, where the board size keeps increasing, etc.

And you can, let's say, similarly define rules that are PSPACE complete. And several games like this, like generalized checkers, have also been shown to be PSPACE complete. Many games that are popularly known have been shown to be PSPACE complete. So that is what I want to say about PSPACE complete. This is something that we already discussed like listing of classes.

And yeah, that's all I wanted to say in this lecture, lecture 60. So basically, two points I wanted to mention that we did not cover in detail or we could not cover in detail in this course but which appears in chapter 8. One is NL equal to co-NL. and PSPACE completeness. So do read the book for the full proofs of PSPACE completeness of these languages like TQBF and generalized geography, etc.

And also for the proof of NL equal to co-NL. Of course, you can also refer to other material if you find other convenient material outside. And with that summary, we come to the end of chapter 8. Right? So we couldn't fully cover chapter 8 because we only devoted one week for chapter 8. But there are other resources available where you can refer and learn.

And so in chapter 8, so far we saw... We saw space complexity, the relation between time and space complexity. We saw NL, we saw L, we saw NL completeness. We saw Savitch's theorem and we saw the summary of these results.

And that completes chapter 8. And that also brings us close to the end of this course. And we will do just one more lecture summarizing what we have seen throughout the course. So, that will also be the last lecture for this course. As far as lecture 60 is concerned, we will end now. So, thank you and see you in the next lecture 61. Thank you.