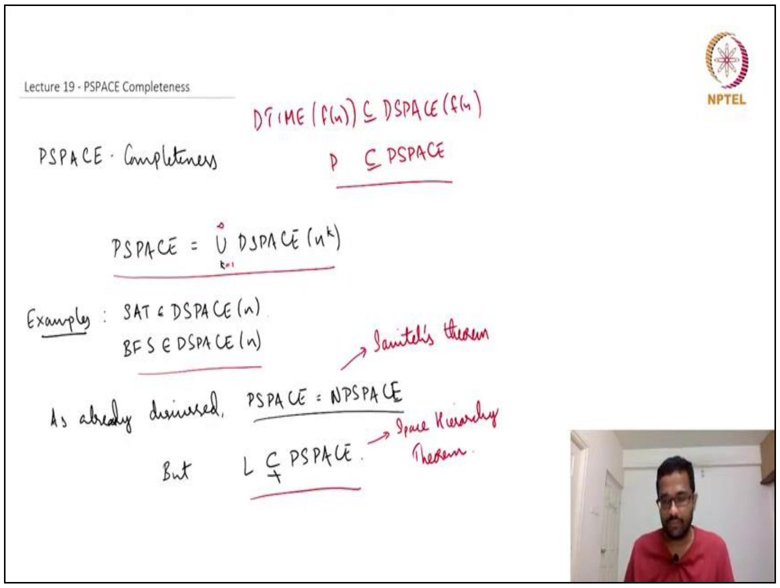
Computational Complexity

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

Lecture -19
PSPACE Completeness

(Refer Slide Time: 00:15)



Lecture 19 - PSPACE Completeness

PSPACE - Completeness

$\mathrm{DTIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n))$

$\underline{P \subseteq \mathrm{PSPACE}}$

$\underline{\mathrm{PSPACE} = \bigcup_{k=1}^{\infty} \mathrm{DSPACE}(n^k)}$

Examples : $\mathrm{SAT} \in \mathrm{DSPACE}(n)$
$\underline{\mathrm{BFS} \in \mathrm{DSPACE}(n)}$

$\nearrow$ Savitch's theorem

As already discussed, $\underline{\mathrm{PSPACE} = \mathrm{NPSPACE}}$

But $\underline{L \subsetneq \mathrm{PSPACE}}$. $\rightarrow$ Space Hierarchy Theorem.

Welcome to lecture 19 of the course Computational Complexity. So, we have been seeing complexity classes based on how much space a certain computational problem takes. So, we have so, far seen classes such as log space L and NL and co NL and so, on. So, today we see another complexity class which is based on the space usage. So, this is PSPACE. So what is PSPACE? It is just a space analog of P.

P was the set of all the languages that could be solved in polynomial time whereas PSPACE is a set of all languages that could be solved in that could be decided in polynomial space. So, it is a union of DSPACE n power k, so, for all integer scale. So, maybe I can say k equal to 1 to infinity. So, anything that can be solved in any polynomial space is is in PSPACE. So, as we have seen just to remind you DSPACE of fn is always contained in d sorry d time of fn is always contained in DSPACE of fn.

Hence P is contained in PSPACE what I am saying is that you we have already seen in one of the previous lectures that the time of fn is contained in the space of the same function maybe I will just move it around here all. So, hence P is contained in PSPACE. So, if something can be done in f n amount of time it can certainly be done in fn amount of space just because in one time step it can write at most one tape location.

So, the space usage will never be more than the time taken anyway this PSPACE is current P is contained in PSPACE this is an easy inference. Some languages that are in PSPACE are let us say SAT for instance.; So, why is SAT is in fact that is linear space why is that because given a particular assignment you could have. So, there are n variables let us say and which means there are 2 power and possible assignments we could have a counter of length n where we cycle through each of these assignments.

And we try out these assignments one by one for the follow Boolean formula and verifying a Boolean formula whether a certain assignment sets it to true or false is not that difficult. So, that doesn't take any significant amount of extra space. So, all the main the main idea is just this. So, the space required is a linear space. Similarly maybe if you consider a breadth first search; given a graph and you want to do the breadth first search it is also linear space.

So, because the graph itself in adjacency list representation reQuires order n + order m where the sorry vertices + edges and whatever is required for the representation of the graph is not it is in the time require the space required for the BFS is linear in that. So, BFS is also DSPACE n. So, this is not that difficult to see. So, I am not going to elaborate further on them and we have already seen that PSPACE is equal to n PSPACE.

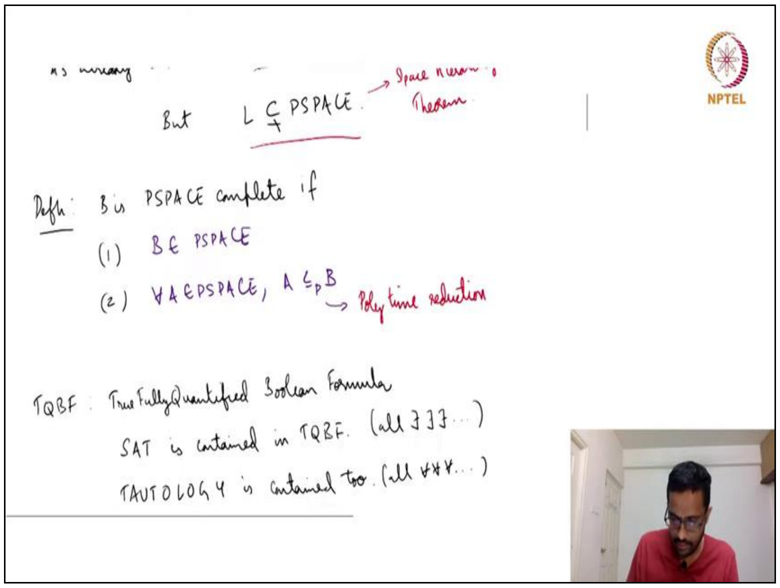
Meaning if I had defined a non-deterministic analog of PSPACE it is the same as PSPACE itself. So, this is something we do not have in time complexity P is different from or we do not know whether P is equal to NP that is a big problem big open problem but whereas in space complexity PSPACE and NPSPACE are the same again this is yet another example of things happening in space differently than how they used to happen in time.

So, why is this was because of Savitch's theorem. So, any polynomial let us say NPSPACE . So, clearly PSPACE is subclass of NPSPACE but then take any non-deterministic polynomial space machine you just square the polynomial that is the space which is a space usage of the corresponding deterministic polynomial space machine. So, it will just be square of a polynomial square polynomial is still a polynomial. So, something takes n power k non deterministic space it can be done in n power 2k deterministic space.

So, if n power k is a polynomial and for 2k is also a polynomial however the class log space is distinct from PSPACE why is that? This is because of something that we are here to see called space hierarchy theorem. So, it is kind of similar to time hierarchy theorem. So, what it says is that. So, time hierarchy theorem kind of said that up to some logarithmic factors if you have 2 functions f 1 and f 2 d time f 1 is strictly contained in d times f 2 where f 1 is smaller than f 2.

Then d times f 1 is strictly smaller contained in d time f 2 meaning there is some language that it can be decided in f 2 time which cannot be decided in f 1 time as long as f 2 is sufficiently bigger than f 1.

(Refer Slide Time: 06:11)



as already

But $L \subsetneq PSPACE$. → Space hierarchy Theorem

Defn: B is PSPACE complete if

(1) $B \in PSPACE$

(2) $\forall A \in PSPACE$, $A \leq_p B$ → Poly time reduction

TQBF: True Fully Quantified Boolean Formula

SAT is contained in TQBF. (all $\exists \exists \exists \ldots$)

TAUTOLOGY is contained too. (all $\forall \forall \forall \ldots$)

Something similar we can say for space also we will soon see this the proof not in this lecture but in an upcoming lecture. So, here we have log space in the left hand side and we have PSPACE in the hand side so that is a distinction. Now let us define. So, again this is just some aspects about space complex PSPACE. So, now let us define PSPACE completeness again this is similar to we have already seen 2 notions of completeness in this course NP completeness as well as NL completeness.

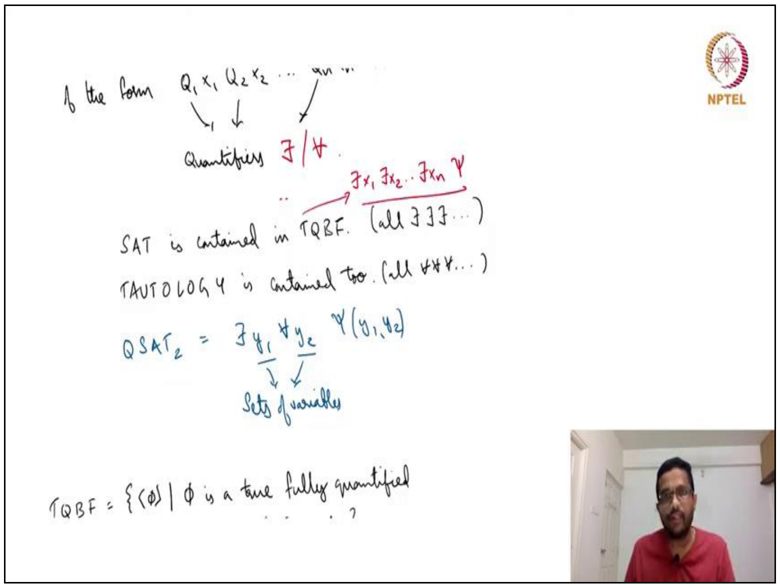
So, this is yet another notion of completeness just like what we have seen before it is it uses reductions uh. So, let me formally define it the first part is simple B is in PSPACE B is PSPACE complete if B is in the PSPACE just like we said B is n and P complete if B is in NP and 2 every PSPACE language every language in PSPACE for all a in PSPACE a is reducible to B but what is the reduction used.

So, in the case of P in the case of NP completeness we had polynomial time reduction in the case of NL completeness we had log space reduction. So, what is the deduction used here. So, one thing to note is that the reduction used the reduction use should not be as powerful as the class itself. So, we cannot say PSPACE reduction here because then it would mean that if any A is in PSPACE if you use PSPACE reduction also then you can you might as well decide that you do not need a reduction to transform it into another thing.

So, the in fact in the case of PSPACE completeness the reduction use is just what we have already seen it is polynomial time reduction. So, as of now we think polynomial time is not it is we do not know if it is as powerful as polynomial space reduction it could be lesser powerful but it could be the same too that is why it is we find polynomial time reduction to be appropriate to use for this requirement.

So, it is something that is that seems to be less powerful than PSPACE. So, again we have already seen peace polynomial time reduction. So, now let us this is PSPACE completeness.

(Refer Slide Time: 10:00)



of the form $Q_1 x_1 Q_2 x_2 \dots Q_n x_n$

Quantifiers $\exists / \forall$.

SAT is contained in TQBF. (all $\exists \exists \exists \dots$) $\exists x_1 \exists x_2 \dots \exists x_n \psi$

TAUTOLOGY is contained too. (all $\forall \forall \forall \dots$)

$QSAT_2 = \exists y_1 \forall y_2 \; \psi(y_1, y_2)$

Sets of variables

TQBF = $\{ \langle \phi \rangle \mid \phi$ is a true fully quantified

Now let us see a a PSPACE complete language again whatever i am saying it is there in Sipser the language is called maybe I will just rewrite this true fully Quantified Boolean formula or it is otherwise called TQBF. So, what does it mean it is a Boolean formula however in the case of satisfiability or in the case of tautology we also had a Boolean formula. But there the reQuirement was different in the case of SAT if I gave you a Boolean formula you had to decide whether there is some assignment that sets its sets the formula to true.

In the case of tautology the Question was is it the case that all the assignments set this Boolean formula to true. So, one is asking is there some assignment that makes it true second is asking do any assignment does it make it true. Both of these can be viewed as fully Quantified Boolean formula. So, what is fully Quantified means? Suppose there is a maybe I will just write this right suppose there is a suppose the n variables there is a formula psi with n variables.

Now a fully Quantified Boolean formula or fully Quantified psi is of the form Q 1 X 1 Q 2 X 2 and so, on Q n X n. So, let us say the n variables are X 1 X 2 up to X n and then psi where each one of these Q 1 Q 2 are they are all Quantifiers. Quantifiers mean they could be existential Quantifier or universal Quantifier there exists or for all. So, basically now what is SAT in this form?

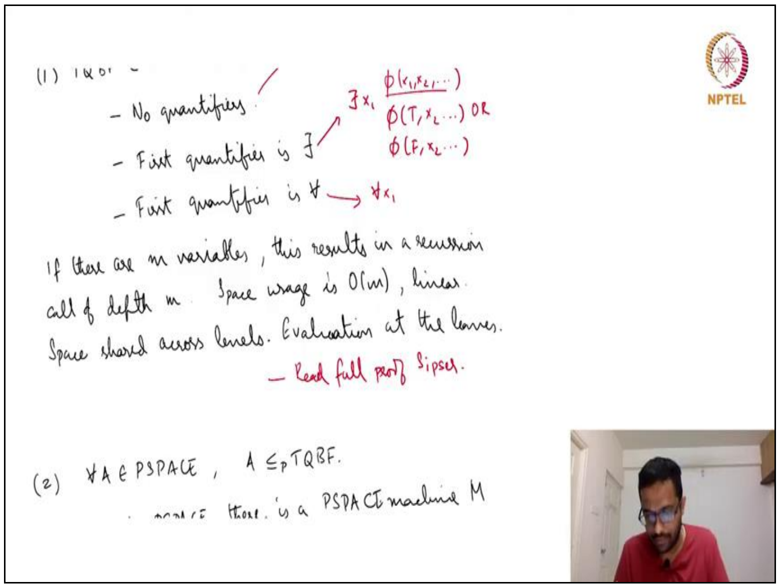
SAT is a fully Quantified Boolean formula where we are asking is does there exist in an assignment such that all of these are said to the formula is set to true. So, we could think of it as where all this Q 1 Q 2 etcetera all of them are existential Quantifiers if all these Q and Q 2 are we are asking there X is X 1 such that there X is X 2 such that there X is X n such that the formula is true that is satisfiability whereas tautology is something where or all the assignments X 1 X 2 etc.

All the assignments for all the variables set to sets the formula to true. So, it is a for all situation maybe I will just. So, SAT is something like this there X is X 1 then X is X 2 that X is X n psi this is set. So, both tautologies as well as satisfiability are TQBF's and if you think about it we saw these languages called Q SAT subscript k. So, they were when we discussed a polynomial hierarchy. So, the square of the form. So, just to just to recollect Q SATs, Q SAT 2 was there exists Y 1 such that for all Y 2 the formula is true.

So, where Y 1 and Y 2 are sets of variables. So, there exists Y 1 such that for all Y 2. So, these are sets of variables some formula psi of Y 1 Y 2 this is true. So, even that Q SAT or sorry this is Q SAT 2 but similarly you can write Q SAT k just that the number of alternations will increase but then all the variables are Quantified. So, Y 1 contains some variables everything that is not there in y1 will be there in y2.

So, Q SAT 2 is in is can be written as a TQBF and hence yeah in anything Q SAT k can also be contained in TQBF. So, Q SAT k or even Q SAT k complement is contained in TQBF they are all they can all be represented as fully Quantified Boolean formula which we are asking whether they are true or not.

(Refer Slide Time: 15:14)


(1) TQBF -

- No quantifiers ✓
- First quantifier is $\exists$ ↗ $\exists x_1$ $\underline{\phi(x_1, x_2, \ldots)}$ $\phi(T, x_2 \ldots)$ OR $\phi(F, x_2 \ldots)$
- First quantifier is $\forall \longrightarrow \forall x_1$

If there are m variables, this results in a recursion call of depth m. Space usage is $O(m)$, linear. Space shared across levels. Evaluation at the leaves.

— Read full proof Sipser.

(2) $\forall A \in$ PSPACE, $A \leq_p$ TQBF.

PSPACE there is a PSPACE machine M

So, yeah formal definition phi is a true fully Quantified Boolean formula but anyway I have explained it and this I have said this before I will say it again the order of the Quantifiers matter. So, it is not like if you swap the order or something else the meaning is retained because of let us say integers for all X does there exist a Y such that Y is greater than X then X and Y come from integers where XY are integers there is no.

So, whatever integer X is I could pick a Y that is bigger than X. So, the first one is true but there is the if you swap the order it does not work because there is no Y that is bigger than all the other all the x. So, there is no integer that is bigger than all the other integers. If you tell me a specific integer i can always produce an integer that is bigger than that but there is no integer that is bigger than all the integers that you can give me.

So, the second one is false again this i have already explained. So TQBF is PSPACE complete why is this? So, let us see why this is the case again why this is the case? So, there could be what are the possibilities? So, one is that there is no Quantifiers but TQBF means all the variables are Quantified which means if there are no Quantifiers there could be no variables. So, this means it is a formula without any variables.

So, formula of only constants if it is a constant formula it is something like the Boolean like true or true and false some something like that some just take a 3 + 4 multiplied by 6 some numerical equation without variables is just like a Boolean equation without variables and one can easily evaluate that without much difficulty. So, this is certainly in PSPACE what if. Now what if there are Quantifiers let us look at the cases. So, look at the first Quantifier.

There are 2 possibilities the first Quantifier is an existential Quantifier there exists or an universal Quantifier for all. So, let us consider both. So, let us the let us say the first Quantifier is there exists Quantifier existential Quantifier. So, it says there exists X 1 and something else some. So, there could be other variables in this something else. Now how do we solve this. So, we want to check whether. So,. Now the formula is saying. So, some formula this formula is saying.

So, this could be psi of X 1 X 2 something or something where the psi also contains Quantifiers or maybe let me just say phi of this where phi also contains Quantifiers there could it will have Quantifiers because. So,. Now we could set X 1 to true and see whether phi is true because everything else is quantified inside phi except X 1. So, if you set x1 to be true it be phi becomes a TQBF instance and you could recursively do the same algorithm if you make x1 to be false again phi becomes a TQBF instance because only variable that is not Quantified in phi is X 1 phi is set X 1 is set to false and the rest are Quantified.

So, again it is a TQBF instance and we want to check whether for a. So, the Question is does that exist in X 1 such that phi is true. So, you want you will check either is phi X 1 is phi true X 2 X 3 etcetera is that true or phi. So, if I true X 2 etcetera is this true or phi false X 2 etcetera is one of these true because both of them are TQBF instances. Now when the first. So, again it is a recursive thing. The second thing is when the first Quantifier is a universal Quantifier again it is not very difficult you could think of it as for all X 1 something.

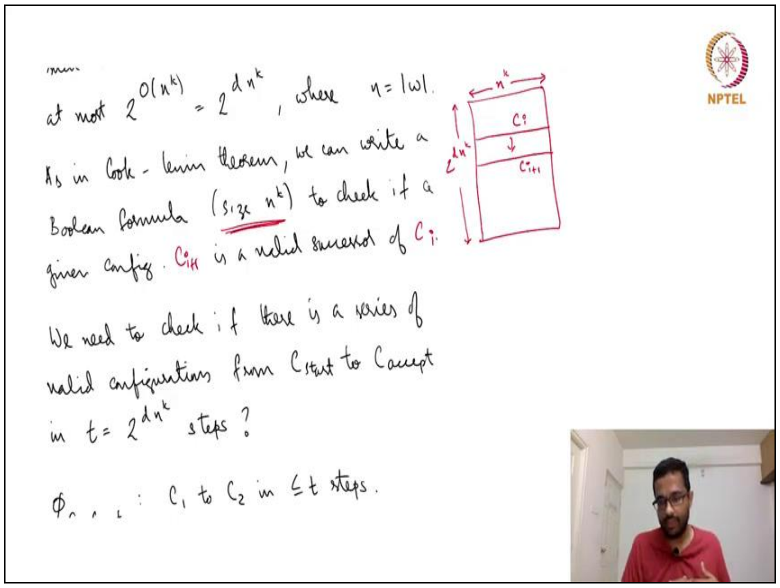
But it is exactly what I described for the universal existential Quantifier except that. Now we want both cases to be true. So, we will check both branches and check if both of them are true. So, what needs to be done here? It is a recursive call. So, at each instance the tentative assignment of phi is X 1 is stored and then in the next level of recursion X 2 will be stored and the next level of recursion the next assignment of X 3 will be stored and so on.

So, if there are m variables m variables in this fully Quantified Boolean formula we need to remember order m variables assignment. So, that is order m, m bits are required and then finally at the end we will just make it we just try to evaluate whether this is true or not. So, we just have an equation of constants a Boolean equation of constants which can be evaluated and then at the end all these branches have to be combined like existential means there should any one of the choices should be true.

So, all this can be done at the base at the top level. So, the space usage is order m because there are m items to be remembered at most. So, this is the high level idea and this proof is there in Sipser read full proof in Sipser. Anyway I explained the full proof here maybe i just did not write it down but it is there in steps or anyway. So, this is the proof that TQBF is in B space.

So, the space usage is order m which is linear space and this space is for all the all those recursion levels not just one level because at each level a constant amount of space is required to store whether X 1 is true or that kind of thing true or false.

(Refer Slide Time: 21:57)



at most $2^{O(n^k)} = 2^{dn^k}$, where $n = |w|$.

As in Cook-Levin theorem, we can write a Boolean formula (size n^k) to check if a given config. C_{i+1} is a valid successor of C_i.

We need to check if there is a series of valid configurations from C_{start} to C_{accept} in $t = 2^{dn^k}$ steps?

$\phi_{c_1, c_2, t}$: C_1 to C_2 in $\leq t$ steps.

n^k
2^{dn^k}
C_i
C_{i+1}

Now let us come out come to the bigger question which is whether it is PSPACE hard just like NP hard we want to show that all the languages in PSPACE are reducible to TQBF. For all A in PSPACE a should be reducible to TQBF. So it might help to recall some details of the Cook-

Levin theorem proof. So, if A is in PSPACE then; so, this is going to be similar to that; but just that we would not go to the details because we have already seen the details in the Cook Levin theorem.

So, we will just appeal to what we did then if A is in PSPACE there is A PSPACE machine that decides a as a PSPACE deterministic machine. Now given a string w we can construct a formula such that w is in A if and only if that formula is in TQBF that is a high level idea of the proof this is how you do reductions. You have you want to show that given a w a string w is in A if and only if some form some fully Quantified Boolean formula is in TQBF this is what we want to show.

So, how do we try to show this. This is the goal of the reduction the second point of the reduction is that the construction of the instance phi should be in polynomial space oh sorry polynomial time because this is a polynomial time reduction. So, the construction of this formula should be in polynomial time. So, how does how is that going to work. So, as I already said there is a PSPACE machine m that decides A. So, w is in A if and only if m accepts w. So, we will use m and w to construct this formula.

So, if n accepts w that means there is a computation table that you can draw we know it is a polynomial space machine. Let us say the space let us say it has single tape let us say the tape usage is in power k. The tape usage is n power k and what is the number of what is the height of this computation table it is a maximum number of rows it can have each row corresponds to a configuration of the machine.

How many configurations can we have before we start if we see the same configuration again it means we are looping which means we are not we have come to the same configuration again; again we will come back to the same thing and we are looping. So, what is the maximum number of distinct configurations that you can see and we have done this calculation many times? It is a maximum number of configurations that we can have for a space bounded machine.

So, if we have an n power k space bounded machine the number of configurations will be 2 power d n times n power k where d is some constant where and n is the length of the input. So, this is like Cook Levin theorem but in Cook Levin theorem it was time bounded. So, the number of rows was n power k and because the number of the time was n power k we said that the row the columns also will be n power the space usage will be n.

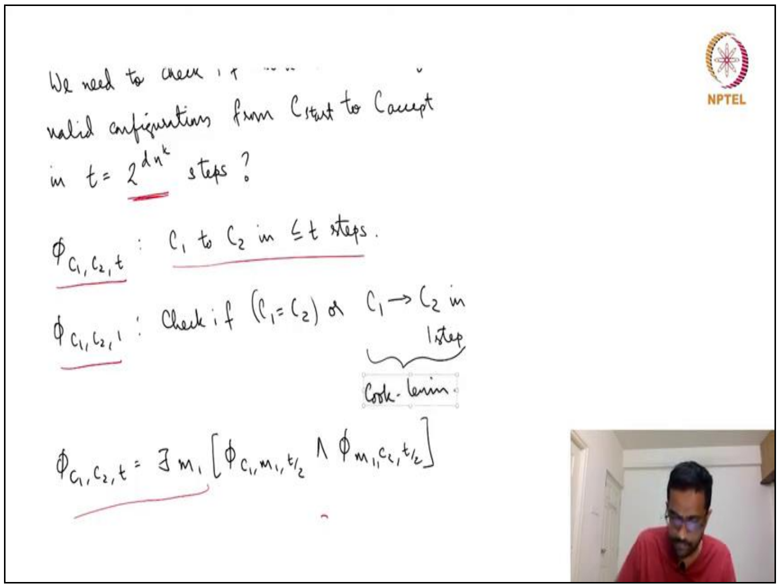
But here it works differently because here we first know that the bound on the; we first know the bound on the number of columns or the space usage and because of the columns are bounded because the columns are bounded the space is bounded we know the bound of the configuration hence the time of the computation will also be bounded in X it will be exponential in this space again. We have seen this d time or DSPACE fn is contained in d time to power order fn. So, the same it is the same thing here.

Now again this is a deterministic machine. So, just like in Cook Levin theorem we can write a formula Boolean formula to check whether c i or c i + 1 is a proper successor of c i sorry a configuration i + 1 is a successor of configuration i this is we can write a Boolean formula for it. And this formula will be of size n power k so given to check whether some configuration c is a or maybe i just make it more clear here c i + 1 is a valid configuration of c i.

So, basically we need to check we know the start configuration given a particular input it is it is known we know the accepting configuration again we can we can do some things to make it we can force the machine to have a unique accepting configuration and we need to know whether there is a path from the start or path meaning a sequence of configuration from the start to the accepting. And we know that the maximum number of steps that we can have is 2 power d and power k steps.

So, can we can this table have a valid sequence of configurations such that the first is the starting configuration last accepting configuration and every configuration is a valid successor of the previous configuration. This is exactly what we did in Cook Liven theorem proof. Just that in this case we are bothered about just that in this case one is that we have a deterministic machine. So, presumably the number of successes are less but that does not really feature in the computation anywhere 2 the number of rows are higher there we have n power k rows here we have 2 power and 2 power d times n power k rules which is exponential in that.

(Refer Slide Time: 28:13)



We need to check if [illegible]

valid configurations from C_{start} to C_{accept}

in $t = \underline{2^{dn^k}}$ steps ?

$\underline{\phi_{C_1, C_2, t}}$: $\underline{C_1 \text{ to } C_2 \text{ in } \leq t \text{ steps}}$.

$\underline{\phi_{C_1, C_2, 1}}$: Check if $(C_1 = C_2)$ or $\underbrace{C_1 \to C_2 \text{ in 1 step}}_{\text{Cook-Levin}}$

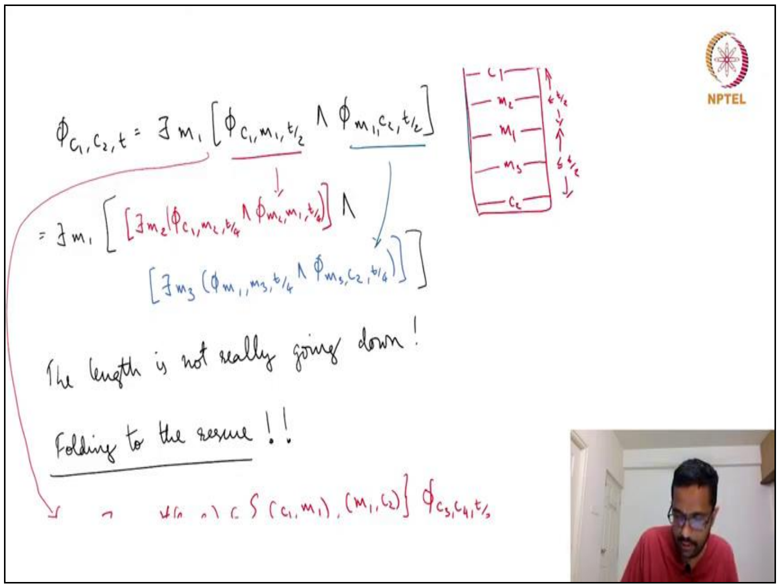
$\phi_{C_1, C_2, t} = \exists m_1 \left[\phi_{C_1, m_1, t/2} \wedge \phi_{m_1, C_2, t/2} \right]$

Now let us write a formula for this. So, let phi c 1 c 2 t. So, what we will do is we will end up a end up writing a fully quantified Boolean formula for this. So, we will use all the Quantifications and that we want. So, the goal is to come up with something generally we want to say does there exist a sequence of configurations that takes you from c 1 to c 2 in t steps. So, now if we can accomplish this then we can set c 1 to c start c 2 to c accept and t to 2 power d times n power k. So, then we can solve whatever we want.

So, let us see what is c 1 c 2 one can we move from c 1 to c 2 in 1 step at most c 1 c 2 one. So, what do you mean by one step at most either the c 1 and c 2 are the same which means zero steps or c 1 and one step computation gives you c 2. So, either c 1 equal to c 2 or c 1 to c 2 in one step and this is not that difficult to do because even equality can be easily checked and c 1 to c 2 in one step again it can be checked by what we saw in the proof of Cook Liven theorem.

We had we had this phi move which was which is essentially this making windows into tables and so on. So, both of this can be done. So, that is fine. So, this formula is of size i think n power k or order n power k this entire thing. So, that is good. Now we want to see whether we can generalize this for not one step what t steps.

(Refer Slide Time: 30:16)



$$\phi_{c_1,c_2,t} = \exists m_1 \left[\underline{\phi_{c_1,m_1,t/2}} \wedge \underline{\phi_{m_1,c_2,t/2}} \right]$$

$$= \exists m_1 \Big[\left[\exists m_2 \left(\phi_{c_1,m_2,t/4} \wedge \phi_{m_2,m_1,t/4} \right) \right] \wedge \left[\exists m_3 \left(\phi_{m_1,m_3,t/4} \wedge \phi_{m_3,c_2,t/4} \right) \right] \Big]$$

The length is not really going down!

<u>Folding to the rescue!!</u>

$$\forall (c_3, c_4) \in \{ (c_1, m_1), (m_1, c_2) \}\ \phi_{c_3,c_4,t/2}$$

So, let us see again we saw Savitch’s theorem let us see how to write this in t steps because if you write in t steps one positive like if we do it in like in Cook Levin theorem we will have to write c 1 c 2 is a successor of c 1 c 3 is successor of c 2 c 4 is successor of 3 3 c 3 and so, on which will mean an exponential set sequence of such things. So, which is 2 power d times n power k times n power k which will be.

So, we cannot afford to do this to check phi c 1 c 2 1 and phi c 2 c 3 1 and c 3 c 4 1 and so on because this will be exponential size. So, we cannot afford to do that. So, we cannot do that. So, how do we do this more intelligently we want to reduce the size of this formula. So, what we will try to do is something that we tried in Savitch’s theorem already we will use recursion. So, if there is a weight from c 1 to c 2 in t steps then there should be a middle point at t by 2 steps t over 2 steps where were you.

Let us say you were in some intermediate configuration called m 1. So, we do not even know what m 1 is but there may be some configuration called m 1. So, I could write does there exist an m 1 such that from c 1 to m 1 you can go in t by 2 steps and m 1 to c 2 you can go in t over 2 steps c 1 to m 1 and t over 2 and m 1 to c 2 in c t over 2. So, basically this is the midpoint of the table, so c 1 here m 1 here which is roughly at the midpoint sorry small m1 and c 2 here.

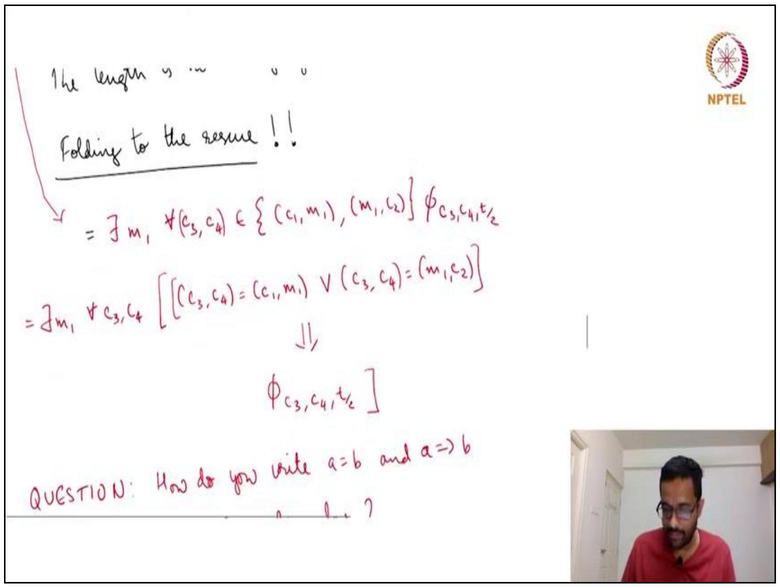
So, this, so, if you can go from c 1 to c 2 in at most t steps then this should be possible right. So, now let us see what can be done. So, now we could we could possibly repeat this trick again. So, there excess m 1 says that this is true. Now i could possibly rewrite I could possibly rewrite this part c 1 m 1 t over 2 into in a similar fashion by this red color stuff and m 1 c 2 t by 2 into this blue colored notation.

So, this here and this here basically between c 1 and m 1 there has to be an intermediate m 2 and between m 1 and c 2 there has to be intermediate m 3. So, we are checking whether c 1 to m 2 can be traversing t by 4, m 2 to m 1 can be traversing t by 4 m 1to m 3 t by 4 and m 3 to c 2 t by 4 that gives you this kind of formula but notice what is happening here I have split the original c 1 c 2 t into 4 parts c 1 m 2 m 2 m 1 m 3 m m 1 m 3 and so, on and the t became t divided by 4 and it is.

So, now the t became 1 4th and we have 4 parts but now this is expanding and expanding and now this does not seem to be and I think even if we try to unroll this t by 4 into single steps this will be actually the original situation again we will have an exponential sized formula without any size reduction. So, this is not going to help. This is just some I am just using recursion but not really making progress here.

So, the length is not decreasing here. So, now so, this is whatever we had in Savitch's theorem does not really help but we now use a new trick called folding. So, what does folding mean? We will see now. So, consider the consider the stop formula again there X is m 1 such that phi is c 1 m 1 t by 2 and phi m 1 c by 2 t by t by 2 c 1 t 2 sorry m phi m 1 c 2 t by 2. Now I am going to write this in a new way.

(Refer Slide Time: 34:49)



The length is ...

Folding to the rescue !!

$$= \exists m_1 \; \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\} \; \phi_{c_3, c_4, t/2}$$

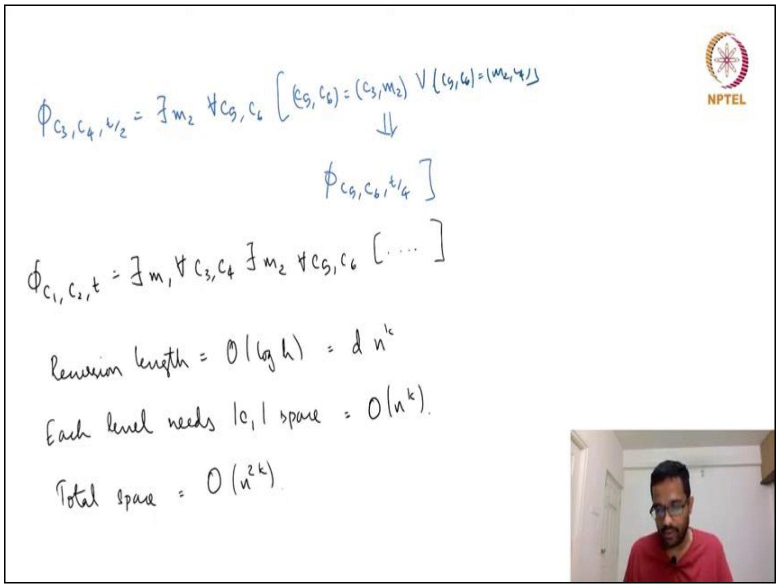
$$= \exists m_1 \; \forall c_3, c_4 \left[[(c_3, c_4) = (c_1, m_1) \vee (c_3, c_4) = (m_1, c_2)] \Downarrow \phi_{c_3, c_4, t/2} \right]$$

QUESTION: How do you write $a = b$ and $a \Rightarrow b$

So, instead of writing phi c 1 m 1 t divided by 2 and phi m 1 c 2 t divided by 2 I will just write it as. So, there X is m 1 here that X is m 1 same thing i will write it as c 1 m 1 and m 1 c 2 are the 2 pairs. So, I will define new 2 variables c 3 and c 4 whenever c 3 c 4 are equal to sorry sorry c 1 m 1 and m 1 c 2 I will check phi c 3 c 4 t by 2. whenever c 3 c 4 or c 1 m 1 and m 1 c 2 I will check phi c 3 c 4 t by 2. So, when c 3 c 4 is c 1 m 1 I am checking phi c 1 m 1 t by 2 which is what we have the first term here and c 3 c 4 is m 1 c 2 I am checking phi m 1 c 2 t by 2. So, for all c 3 and c 4 such that they are like this they are in c 1 m 1 or m 1 c 2 I do this.

So, I am using their axis in m 1 and for all c 3 c 4 in this set. So, now I am using some new kind of expressions here. So, I am using contains is a member of c 1 c 1 m 1 how do we write that in a Boolean setting it is not very difficult.

(Refer Slide Time: 36:38)



$$\phi_{c_3,c_4,t/2} = \exists m_2 \forall c_5, c_6 \left[(c_5,c_6) = (c_3,m_2) \vee (c_5,c_6) = (m_2,c_4) \Downarrow \phi_{c_5,c_6,t/4} \right]$$

$$\phi_{c_1,c_2,t} = \exists m_1 \forall c_3, c_4 \exists m_2 \forall c_5, c_6 [\ldots]$$

Recursion length = $O(\log h) = d\,n^k$

Each level needs $|c_1|$ space $= O(n^k)$.

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

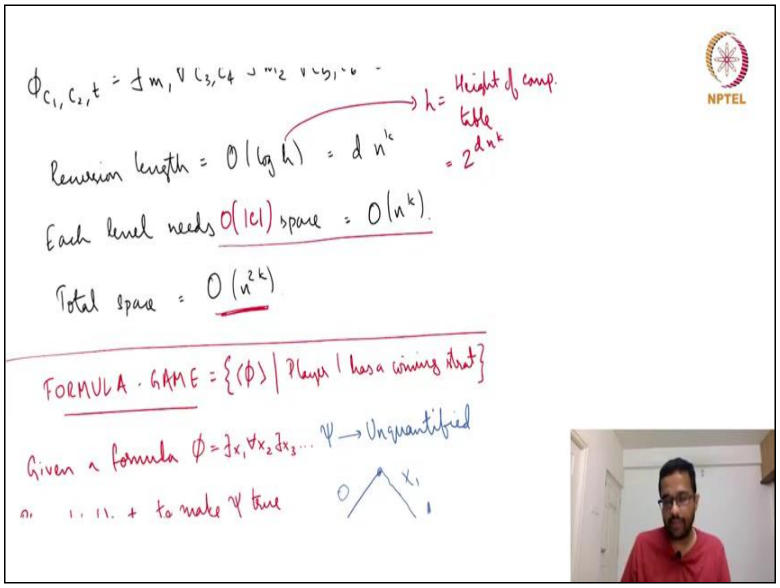
So, what you do is you can have it this particular part this blue underline part can be rewritten by this part sorry can be rewritten by this entire thing. So, whenever c 3 c 4 is equal to c 1 m 1 or c 3 c 4 is equal to m 1 c 2. So, this m 1 c 2 this kind of looks like an e but it is c 2 when this happens this implies that phi c 3 c 4 t divided by 2 must be true. So, the rest is same phi m 1 for all c 3 c 4 whenever c 3 c 4 is equal to c 1 m 1 or c 3 c 4 is m 1 c 2 phi c 3 c 4 t by 2. So, now the first part is the only thing that is remaining is the equal to symbol and the implies symbol.

Now finally I leave it to you to think how to write sorry how to write the equal to symbol and the implies a implies b and a equal to b how do you write it in a in a using just Boolean formulas these are very simple exercises maybe you might have seen this in the first class in Boolean logic. So, you can think about it this is not very difficult. So, you can the point I want to impress upon you is that all this can be done in all this can be done in using Boolean formulas.

So, now the formula is something like this. So, this is this entire formula is now some constant multiplied by n power k the part after the quantifiers . Now again phi c 3 c 4 t by 2 is a is something that we need to go deeper into. So, how do we go deeper into that? Again we will use exactly the same thing I can define intermediate states c 5 and c 6 and the intermediate state m 2 and then variable c 5 and c 6 such that maybe I will just write it with c 5 c 6 is equal to c 3 m 2 or c 5 c 6 is equal to m 2 sorry sorry is equal to m 2 c 4 and even that happens this implies phi c 5 c 6 t divided by 4.

So, this is how you expand out this quantity the blue underline Quantity phi c 3 c 4 t divided by 2.. So, now when I combine all this. So, recall we were trying to write a formula for phi c 1 c 2 t which we first noticed as phi that there exist m 1 such that for all c 3 c 4 something and the rest. So, I can write it as there exist m 1 such that for all c 3 c 4 there exists m 2 such that there are faults c 5 c 6 there will be some formulas involving c 3 c 4 c 1 m 1 all this and c 5 c 6 c 3 all this and then finally we will have phi c phi c 6 c 4 as part of this which again can we go deeper and deeper into.

(Refer Slide Time: 40:16)



$\phi_{c_1, c_2, t} = \exists m_1 \forall c_3, c_4 \exists m_2 \forall c_5, c_6 \ldots$

Recursion length $= O(\log h) = d\, n^k$ $\quad h =$ Height of comp. table $= 2^{d n^k}$

Each level needs $\underline{O(|c|) \text{ space } = O(n^k)}$.

Total space $= \underline{O(n^{2k})}$

$\underline{\text{FORMULA}\cdot\text{GAME}} = \{\langle\phi\rangle \mid$ Player 1 has a winning strat$\}$

Given a formula $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots \psi \rightarrow$ Unquantified

to make ψ true

So, the point is that each level adds only a linear some constant multiplied by n power k which is each this is what I have written here in the part that I am underlying. Now each level needs extra spaces some constant times order. So, so order size of c means order size of a configuration space and we know the size of a configuration is order n power k. So, it is it is still another constant. So, it is still order n power k this is the extent of memory required at each level and at each level notice that the t became t by 2 and then t by 4.

So, every level that the t is halving and we know what to do at the base case when t equal to 1 this is what we could do this is what we did. So, the base case is at 1. So, how many levels are required to go from t to t divided by 2 to t divided by 4? So it is log t number of levels. So, or the number. So, the num height of the ah. So, I am saying log h here where h is the height of the computational table which is 2 power d times n power k. So, log h is the height of the computational table.

So, where maybe I will write that also h is equal to height of computational table which is equal to power d times n power k. So, I am going to take log of that it is just d times n power k. So, this is the number of levels and at each level you require order n power k space. So, the total space used is some constant times d power k sorry n power k multiplied by order n power k which is order n power 2 k multi with a constant. So, it is order n power 2k and this is the total space required to write down the formula.

So, this means that this is the total space required to for the for the computation of the formula and even to write down the formula it is a similar amount of space. So, this is a recursion length and this is the number of space required. So, hence and the correctness is kind of evident because again we from our experience in the Cook Levin theorem this formula captures whether there is a there is a sequence of configuration that need a string to be accepted which happens when the string is accepted by a machine which happens when the string is in the language.

So, again everything is if and only if. So, the correctness of the reduction is evident this the the time complexity is total spaces and power 2k and the time taken for the computations also this is a very structured thing that is also is polynomial time. So, just to summarize this we defined what is PSPACE which is polynomial it is a space analog of P. We defined PSPACE completeness again with space analog of kind of a space analog of NP completeness.

And the first PSPACE complete language that we saw was TQBF which is short for true fully Quantified Boolean formula where all the n variables are Quantified we saw that SAT Q SAT 2 tautology everything is in TQBF everything is a sub problem of TQBF and we saw Y TQBF is PSPACE complete the fact that please TQBF in PSPACE is it is rather straightforward.

And the proof that any P's based PSPACE language can be reduced to TQBF requires us to go through computation table and some simplifications in the Boolean formula that encodes the configuration. And that is that is about it for this lecture I will complete the rest of the PSPACE completeness in the sec in the next lecture. I just wanted to add one more thing. So, notice that we are doing this under polynomial time reductions and we saw that SAT is contained in TQBF.

Tautology is can be represented as a TQBF instance Q set 2 Q SAT k Q side k complement all of them are can be represented as a TQBF instance this means that NP satisfy an NP complete language all of NP can be is contained in PSPACE. All of co NP technology is a co NP complete language is contained in PSPACE all of sigma 2 is contained in PSPACE sigma k is contained in PSPACE, pi k is contains PSPACE.

So, the entire polynomial hierarchy is contained in PSPACE this is what it means. So, think about this particular fact. So all of NP co NP sigma 2 sigma k pi k all are contained in PSPACE and with that thought I think I will close this lecture in the next lecture we will see a couple of more instances of instances of PSPACE complete languages, thank you.