[SQUEAKING]

[RUSTLING]

[CLICKING]

MICHAEL SIPSER: Let's get back to space complexity. So just to review what we've been doing from last lecture, which feels like a long time ago but it was only two days ago, we were looking at those three theorems, which all had basically the same proof. One was about the ladder problem, where you're trying to see if you can get from one string to another string changing one symbol at a time. And all of the strings were in the language of some finite automaton, or you had some reasonable rule for saying which are the valid strings, which are not.

And then we built on that idea. We showed Savitch's theorem, that going from any nondeterministic machine to a deterministic machine only squares the amount of space. And then we finally proved that TQBF, this problem about quantified Boolean formulas, testing whether they are true, that that problem is complete for PSPACE.

So we're going to build off of that last theorem today, and talk about one other complete problem, and also show a connection between PSPACE and testing which side can win in certain kinds of games. And then at the end of the lecture, the second half of the lecture, we'll talk about diving deeper into space complexity. We're going to talk about a different part of the parameter space where instead of looking at polynomial space, we're going to talk about what happens if you have logarithmic space, which is another natural point to look at for space complexity. But we'll get to that after the break.

Talk about game complexity and games-- so when I was little, we used to play, me and my sisters, would play a game called Geography. I don't know how many of you have heard it or maybe it goes under a different name. But it was a game basically about words and places.

And it has a very simple set of rules. Let's say there were two people. You take turns picking names of places.

And then each person has to pick a place which starts with the letter that the previous person's place ended with. So for example, you have two people playing. Perhaps you agree on some starting place like a major city nearby, like Boston.

So one player, the first player, picks Boston. And then after that, the next player has to pick a place that starts with N, because Boston ends with N. And so maybe Nebraska would be a possible response to Boston.

And then the first player would have to respond to that with a place that starts with an A, because Nebraska ends with an A, maybe Alaska, which also starts with A, ends with A. So then Arkansas, maybe, would be a reasonable response to that. And then that ends with S, so San Francisco, and then so on, and so on.

And of course, you want to forbid people from reusing names because then you'll just get into a loop. Saying, "Alaska," "Alaska," "Alaska," won't make the game very interesting, so you have to forbid that possibility. Names cannot be reused. They can be used at most once.

And then, eventually, one side or the other is going to run out of names and not have-- either because of the lack of their geographical knowledge or maybe you've just exhausted all the possibilities-- and will not have a good response. And that person will be stuck. And that person is the loser. The other person has won the game. So the objective is to try to avoid getting stuck.

So let's just take a look. I'm going to think about that game in a slightly more abstracted, formalized way. So you write down all of the legitimate places. They become nodes of a graph such as I've shown here. And then you draw an edge from one node to another if that corresponds to a legal move in the game, a legal step of the game.

So Boston connects to places that start with N, like New York and Nebraska. So the first person might pick Boston. The second person might pick either of these two, and then back to the first person. If they picked Nebraska, they could pick Arkansas. They can pick Alaska, and so on, just as I described.

Believe it or not, we actually played this game. I can remember playing this game when I was a kid. That was, of course, before we had "League of Legends" and other fun stuff, but this is what kids used to do.

So anyway, just to get it down on the slide, the rules are-- assuming we have two players, we'll call them Player I and Player II. Player I goes first. And they take turns picking places that start with the letter which ended the previous place, no repeats allowed. And the first player stuck loses, the person who doesn't have a move to make.

So what we're going to look at today is a generalized version of that, where we're just going to take away the names, but just leave the underlying graph. And we call it going to call that "generalized geography." So here, that can be played on any directed graph.

You take away the names of the nodes. Now you just have some arbitrary graph. You're going to designate a particular node as the starting node.

And then the players take turns following those edges. And because you're going to forbid having any loops reusing any nodes, what you end up doing is, working together, you're going to be constructing a simple path in that graph. Simple path just follows those edges and never intersects itself.

And the first player to be stuck is the loser. So the other one is the winner. So we're going to call that "generalized geography."

And we're going to look at the computational complexity of deciding, for a given graph and starting point, which side would win if both players played optimally, the best possible. And we'll name that problem GG, or look at its associated language. So here is a graph and the starting node.

And you want to say that pair is in the language GG if the first player, Player I, who must play at that node A to start off with has a forced win in that generalized geography game on that graph starting at A. Again, what I mean by a forced win-- I'm not going to define this formally, though we could do that-- it's not hard to do.

But a forced win is also called a "winning strategy." That just means that if both sides play optimally, they play the best possible-- not of their capability, I mean absolutely the best possible play-- then that Player I would still win. There's nothing that Player II can do to prevent Player I from winning if Player I has a winning strategy or a forced win.

You may have seen, for example-- there are examples of this that-- I mean, I don't even know if they still exist, but it used to be in the newspaper. There used to be examples of chess boards and they would start from a certain position. And they would say, "White to win in three."

And so that means white has a forced win. No matter what black does, white is going to end up winning. But you have to think of what the strategy is. It may not be so obvious to think through what are the right moves that white makes. But the point is that no matter what black would do, white would end up winning.

And what you can show-- we're not going to do that-- but for a class of games that includes this generalized geography, either one side or the other is going to be guaranteed to have a forced win. That's not necessarily true for all possible games, obviously, but for a large class of games, one side or the other is guaranteed to have a winning strategy or a forced win. So let's just review that, because this is going to be essential for the first half of the lecture-- to understand what we mean by a game and what we mean by one side or the other to have a forced win.

And what we're going to show is that GG is PSPACE complete, that the problem of given one of these graphs and a starting point, figuring out does Player I have a forced win or is it Player II that has a forced win? Which side is guaranteed to win under optimal play? That's a PSPACE complete problem. And so let's do a little checking on that.

And so I'm going to give you a very small example of a generalized geography game. You have to figure out whether it's Player I or Player II that has the forced win, the winning strategy, or maybe neither, or both. Those are the four options.

And you understand Player I has to play here, because that's the starting place of this generalized geography game. So then it's up to Player II to continue on. All right, so let's launch that poll. You'll have to think about it a little bit, and let me know what you think.

Which side has a forced win? Which player makes the first move? It's Player I makes the first move, and Player I plays, as I showed you here, this is Player I playing at A. So Player I picks A, and then Player II has to pick one of these two.

All right, I think we're the end of the poll. Everybody's in? Pick something. You guys did not do well on this poll. At least not too many of you picked D. That's a little reassuring because you can't have both players having a forced win.

I did say that in games of this kind, one side or the other is going to have a forced win, so C is not a very good choice, either. So maybe you should spend less time reading your email and more time listening to the lecture. The actual correct answer is B.

Player II, which obviously is a minority position here-- Player II has the forced win. Let's understand why. So Player I plays here. As I said, that's the first move.

Player II can either take the upper node or the lower node. Player II takes the upper node, then Player I has no choice but to take the right-most node, and now there's no more move for Player II. So Player II will lose if Player II takes the upper node, the upper choice there. Because Player II will go here, Player I will go there, and it'll be game over. And Player I will have won because Player II was stuck.

However, Player II had another choice. Player II could have also gone down here. So Player II, if it played down here, things are looking a lot better for Player II. If Player II goes here, Player I's move is forced, goes here.

But now, there's another move left that Player II could make. So Player II goes here, I goes there, Player II goes up there. And now Player I is stuck.

So it's Player II's choice, Player II can make a move which will end up causing Player I to get stuck, because that's the nature of what we mean by having a winning strategy. There is some way to guarantee if you're playing optimally, and you're certainly playing optimally, means you'll take that lower move.

Then you will take that. So under optimal play, Player II will win this game, and so Player II has a winning strategy. It has a forced win.

So I see. So I'm not going to define "playing optimally." There's some questions here about that. I'm going to-- I'm going to leave it somewhat informal, but we could make that all precise. I don't think it would be clarifying to go through a precise definition of that.

We don't even have to think about it in terms of optimal play. You can say that a player has a forced win if they have moves that they could make in response to every possible move the opponent could make that will guarantee that they will end up winning. So you don't even have to talk about optimality here. Just talk about every possible opponent. Every possible opponent is going to lose to somebody who has a winning strategy and follows that strategy.

I can see there was some confusion about who was going first and so on. I'm glad. That was part of the reason I gave this check, just to hopefully to clear up some of those points. So Player I plays here then Player II goes.

And I got some comments that some people were confused about what does it mean where does the first person move. Hopefully, that's clearer. All right, so let's move on because we're going to be spending a while proving something about generalized geography in these graphs.

So let's continue. So in order to make the connection between games and complexity, we're going to need to talk about a problem, one related to one that we've already seen, where we can reformulate that problem as a game. And that's going to be a problem on quantified Boolean formulas called the "Formula Game." So this is a different game now, not one that you're likely to end up playing.

But in a sense in which you'll see, this is still a reasonable game where two players can play against one another. And one of them is going to end up winning and one of them is going to end up having-- one or the other will have a winning strategy. So let's understand what the game is.

The game is played on a formula. So you write down a quantified Boolean formula. Remember, we talked about that for several lectures now. It's a bunch of quanti-- variables with quantifiers.

And then there's a part that has no quantifiers, which you can always put into conjunctive normal form if you want. But for the purposes of this discussion so far, it doesn't matter. Later on, we'll actually want it to be in CNF. But for now, we just have some formula.

And associated to that formula there is a game, in a very natural sense. So first of all, there are going to be two players. But now the names of those players are going to be Player Exist and Player For All.

And this is how the game goes-- both players are going to be picking values for variables. They're going to say variable x1 is true, variable x2 is false, variable x3 is false, variable x4 is true. And that's how the moves of the game go. They're just assigning values to variables so that in the end, all of the variables have been assigned a Boolean value. And so we end up with an assignment.

But before we get to that stage, which players get to pick which variables? And the way it works is the Exist player is going to assign values to the Exist variables. So Exist will pick x1 and x3 and assign them values. The For All player is going to assign values to the variables that have a For All quantifier attached. So in this case, the For All player will pick the value for the x2 variable. Fading out here.

And the order of play is according to the order of the appearance of the qualifiers in the formula. So the way I have it written out in this particular formula, it's Exist x1-- so Exist will pick the value for x1. Then For All's turn. For All will pick the value of x2. Then it's going to be Exist picking the value for x3, and so on, until they get to the end, and all of the variables have been assigned to value by one side or the other. And now the game is over.

So what's left is to understand who has won at that point. And the way we say who has won is Exist wins if in the end, this part psi, which is the part without quantifiers, ended up being satisfied by that assignment that the two players together ended up picking. And then For All will win if that part of the formula is not satisfied.

So let me just write that down here. After the variables have been assigned values, we determine the winner. Exist wins if the assignment that built cooperatively together satisfied psi, and otherwise For All, if it doesn't satisfy psi.

So think about it this way-- Exist is picking the values of those variables. He's trying to make this part of the formula satisfied, trying to satisfy this part of the formula. For All is picking values, but she's trying to make this part of the formula unsatisfied. So they're in opposition to one another.

And in the end, one of them is going to have succeeded and the other one will have failed. And now, thinking about it as a game, depending upon the formula, one side or the other is going to have the ability to dominate the other one and always win. And so the question is, which side has that ability to always win? Which side has the winning strategy?

Of course, it'll depend upon the formula. Some formulas, it'll be the Exist player who has the winning strategy. And other formulas, it will be the For All player who has the winning strategy.

And computationally, we'd like to make that determination. Which side is going to win for a given formula? And there's a very, very nice, and actually very easy and simple way to understand that because we've already run into that problem before.

The problem of determining which side has a forced win is exactly the same thing as determining whether this quantified Boolean formula is true because the Exist player has a forced win when that formula-- let me say it again-- the Exist player has a forced win when that formula is true. So if this quantified Boolean formula were a true quantified Boolean formula, then Exist is guaranteed to be able to win that game if it plays its hand right. If this is a false formula, the For All player will always be able to win if it plays its hand right.

So why is that? It really follows kind of almost without doing any work at all. This proof, even though it looks superficially like it might be not so easy to prove, it follows for a very simple reason because the meaning of what it means for Exist to have a winning strategy is exactly captured by the semantics of the quantifiers.

Let's see what that means. What does it mean that Exist has a forced win, let's say up here? That means that Exist has a move that it could make, that Exist can pick some value for x1-- so there was some way to assign x1 so that no matter what the opponent does, no matter what For All does, there's going to be a response that Exist can make, some assignment to x3, so that no matter what For All does, there's going to be an assignment to the next variable, and so on and so on, so that this thing ends up being true.

So let me just say that again somehow. I didn't feel it came out very clear. So Exist has a winning strategy exactly when there is some assignment to x1 that Exist can make such that no matter what assignment that For All makes to x2, there is some assignment x3 such that no matter what assignment that For All makes to x4 and so on, makes this thing true. That's what it means for Exist to have a winning strategy.

That's exactly what the quantifiers are saying in the first place. So even really without doing anything, you can see that "Exist has a winning strategy" means exactly the same thing as the quantified Boolean formula being true. So making the test of which side has a winning strategy is the same as testing whether the formula is true.

So let me try to turn to the chat before we move on to anything else. Here we go. So I'm getting some questions about the order of play and about the alternation of the quantifiers here.

So the way I'm showing it is that the quantifiers always alternate-- Exist, For All, Exist, For All. That doesn't really matter. You can have several Exists in a row, and that would correspond to Exist making several moves in a row before turning it over to For All.

But alternatively, we could just add extra variables which don't play any role in psi. And they're just kind of dummy variables that serve as spacers so if you have two Exists in a row and you don't like that, you can just add a For All of some variables that you never see again in between. So you can always get it to be alternating if you want.

OK, also question-- the difference between phi and psi. So psi is the part here that does not have any quantifiers. So the way we always write down our QBFs is they're leading quantifiers with the variables, and all variables have to be within the scope of one of the quantified variables. So there are no free variables that are unquantified in our QBFs.

And so all of these QBFs have a run of quantifiers and then a part without quantifiers, so the Boolean logic part. Psi here is the Boolean logic part. And phi is the whole thing together with the quantifiers.

So I'm not sure I understand some of these-- why not have the For All player have a forced win? The For All player might have a forced win. One side or the other is guaranteed to have a forced win. So if Exist does not have a forced win, then For All will have a forced win. I'm not going to prove that. That's a fairly simple proof by induction, which I'm not going to go through. Just take it as a fact.

Somebody is asking, why does For All want to not satisfy this expression, the psi part? Well, that's the way we set up the game. In order to make this correspond to TQBF, just the truth of this expression, we want to make Exist try to satisfy this part and For All try to make it not satisfied because that's what works. I'm not sure what else, what other why I can answer there.

I'm going to answer some of the most basic questions here. Somebody is asking, how do we know how many variables to use? So the variables are the variables of the formula.

So somebody is going to hand you a formula. That's going to have x1 to x10, whatever that formula has, some number of variables. And so that game is going to have 10 moves.

If the quantifiers alternate, then the moves will alternate, but whatever the pattern of the quantifiers is going to be the pattern of the moves. So Exist is going to follow the places where the Exist quantifier occurs, and For All is going to follow the places where the For All quantifier occurs. So if you have Exist, For All, Exist, For All, Exist will move first, then For All will move, then Exist will move, then For All will move.

Together, they're picking values to their respective variables with their opposing goals. Exist is trying to make the psi part satisfied. For All is trying to avoid making it satisfied, to make the assignment not satisfy that part. And if you think about what that means, just in terms of the meaning of that, it's going to mean exactly the same thing, that the formula is true.

So anyway, if you don't-- let's see. I don't know if there's any more questions that I can try to answer here. It's still about the numbers of variables. Each formula is going to define a different game. You see the formula.

So the formula, however big-- it may have 50 variables, it may have two variables. Well, we'll do an example. Maybe that'll help. The next check in, which is coming up pretty soon-- might actually be now-- we'll give you an example. So we'll see who is understanding and who is not.

So let's continue here. So therefore, the problem of determining does Exist have a forced win is exactly the TQBF problem, because Exist has a forced win exactly when the formula is true. And what we're going to show is that TQBF is a polynomial time reducible to generalized geography, but conceptually, we're going to think about TQBF now as a game because that's how generalized geography is set up.

So given a formula like this, we're going to construct an instance of generalized geography. We're going to construct a graph where play on that graph is going to mimic play in the formula game. So making the moves in that graph are going to correspond to setting variables true and false because the graph is going to be specially designed to force that behavior.

So I think we have a check-in coming in. Yeah, so let's just see how-- I suggest you pay attention to this check-in, really try to think about it and solve it, because I think that'll help you make the connection between the truth of the formula and Exist having a winning strategy. So if you take this formula here-- this is some particular formula, Exist x for all y, blah, blah, blah.

There's going to be associated to that formula, a formula game where first Exist moves, assigns a value for x, and depending upon that value, For All is going to move and is going to assign a value for y. And after that's done, this part here is either going to be true or false, is either going to be satisfied or unsatisfied. And I want to know, can Exist always find a way to guarantee that this part is going to be satisfied, or can For All always find a way to guarantee that this part is not satisfied? So you have to look at this formula to understand which side is going to end up succeeding.

So if you're not clear on the rules, look back here. Exist is trying to make this part satisfied. For All is trying to make this part unsatisfied. But they neither of them has the totally upper hand, because Exist is picking one of the variables, For All is picking the other variable.

But they're doing it in a certain order. First Exist is going to pick, then For All is going to pick. So that's the way the game works-- Exist picks a value, then For All picks a value, and then you see who wins.

So who wins? Who's guaranteed to win? Can we make sure that Exists always wins or can we make sure that For All always wins for this particular formula? So there's just two variables here. You can think about this in either of two ways-- strictly speaking, purely as a game, or you can look at, understand whether that formula is true or not.

They're equivalent ways of looking at the problem. In fact, in a certain sense, it's the same. That's what I'm trying to get across. Thinking about this as a game or thinking about it in terms of quantifiers-- it doesn't make any difference.

Almost done here. So all right, last call. Closing the poll. I think this one you got. You did pretty well on.

So the correct answer is the For All player has a winning strategy, has a forced win. And also, similarly, the expression, phi is false. Because if you try to find some x such that no matter what y you pick, this is going to be true, this is going to be satisfied, you're out of luck.

Because if you make x true, well, then-- now I'm confused. What is going on. If you make x true, then x bar is false, so y bar has to be true. Now I'm completely confused. Oh, no.

If you make x true, you have to work For All y. So this thing is clearly false. If you make x true, then this x bar is going to be false, and it's not the case that for every y, setting for y-- because y is forced here. y is going to have to be true. y bar is going to have to be true, so y is going to have to be false.

So if you try again to make x false-- maybe that's one to think about first-- if you make x false, you're going to be stuck on the first clause because it has to work for both settings of y. But maybe thinking about it as a game is better. No matter what x does, y is going to have a way of making one of those two clauses unsatisfied.

So if x is true, we can set y to make the second clause unsatisfied, and if x is false, y can be assigned to make the first clause unsatisfied. So anyway, the right answer is the For All player has the winning strategy here. And at the end of the day, it's not critical that you understand this correspondence, but you have to then, if you don't quite understand that, you're going to have to take on faith that the formula game is the same as TQBF because that's what we're going to use.

And by the way, I would like to also mention here that this correspondence between games and quantifiers is something that mathematicians use all the time. Because if you have some expression which comes up often in mathematics, where there are a whole run of quantifiers in front of some statement, it's really kind of hard for anybody to really get a feeling for what that means. If you have six alternating quantifiers, which happens often in mathematics-- you're going to have statements that have a lot of alternations with quantifiers-- very hard to get a feeling for what that means.

But if you think about it as a game, it's much more intuitive. And it's completely equivalent to think about quantifiers. Go back and forth between quantifiers and games.

Anyway, let's look at the reduction, the construction that shows generalized geography is complete. A little longer than I thought-- I just want to make sure everybody is with me on this. So why don't we call the break now, and then we'll come back and we'll look at the construction after that.

Because the construction itself is going to take about 10 minutes to work through to, figure out how to reduce TQBF and build a graph that simulates the formula. So I'm going to move right on to the-- and we'll come back. So feel free to ask some questions and get ready for us diving in to a construction for the reduction of TQBF to generalized geography.

So it is an interesting question about what's the relationship between a formula and when you swap the order of the quantifiers. The questioner is asking, does that somehow relate to negating the quantifier-free part, the part without the quantifiers? And I don't think that that's the right correspondence.

There actually is some relationship. When you say "there exists for all," it's actually a stronger statement than saying that "for all there exists." And in general, that actually implies-- exist x for a y implies for a y there exists an x, because what you're saying is that if there exists an x for a y, there is one x that works for every one of the y's.

If you're saying for a y there exists an x, the choice of that x can depend on the y. So there is a connection, but you have to think through what it means in order to understand that connection. You won't need to know that in order to process what we're going to be doing, but maybe it just helps you think about it.

Other questions here. I think we're out of time here. So let's return to our lecture.

So I'm going to go back. Hopefully that won't crash everything. There we go.

So now we're going to be reducing TQBF to generalized geography. Are we all together now? OK. So I'm going to illustrate this construction, which is a very nice construction, by the way.

I'm going to illustrate this construction just by doing an example, or a partial example, but I think it'll give you the idea. So you understand what I'm trying to do here, is I'm starting off with a quantified Boolean formula. So here it is.

I'm going to assume it's in conjunctive normal form, which I can always convert it into that form, maybe by adding some additional variables, but without doing anything too drastic to it. And so now, starting from that formula, I'm going to now build a graph, where playing the geography game on that graph-- which is, remember, taking turns, picking nodes which form a path-- is going to correspond to playing the formula game, which is picking the variables of that formula. And then you want-- if the Exist player wins in the formula, then Player I is supposed to win in the geography game. So here is how the graph is going to look. So try to follow this.

So there's going to be kind of two parts, one that's going to correspond to the variables, and the other part that's going to correspond to the clauses. And so for each variable, I'm going to have a diamond structure here. So there's a little starting thing that's going to be unique to the first variable.

But then every variable is going to have a little diamond structure here. And they're going to be attached one to the next. And we'll understand what this means when we play geography on this piece of the graph, but let's just understand the structure first.

So this is the start node. And now, here Player I is going to play the role of Exist. Player II is going to play the role of For All.

And in fact, I'm going to identify that. So I'm going to call-- because it's going to be helpful just to think about Player I as being the Exist player. So the Exist player is going to be playing on this graph. The Exist and the For All players are going to be playing on the graph. It's really just Players I and II.

So the Exist player, Player I, by the rules, has to pick the start node. And now it's Player II's move, the For All player's move. Now, if you're with me on this graph, the For All player's move is now not very interesting because it's forced. It has to go to here, because again, they're just picking the nodes of a simple path in this graph. So the For All, Player II, the For All player, has to go here if Exist started over there.

Now it's the Exist player's turn. And now something interesting happens. The Exist player has a choice-- it can either go left or right. There are two possibilities.

And then after that, the For All player's turn, who is, again, just forced. So far, For All has not had much interesting stuff to do in this, has not had to think hard. So no matter whether Exist went left or right, the For All player's move is forced and ends up over here.

And now it's this player's turn. And again, this is kind of an easy one. Oh, before I do that, let's just look at how that play could go.

So the first two moves, as kind of I was suggesting, Exist goes here and then For All goes there. I'm going to illustrate possible plays through this graph by tracing them out in green. So now Exist goes here, For All goes there.

Those were forced. But now, the Exist player goes, could either go this way or could go that way. So those are two possible different ways of playing the game that we're building.

And now, whose turn is it here? This was the For All player picked this one, so the next turn is the Exist player, and that's forced. But now notice-- now for the first time, the For All player has to think, because the For All player can either go left or right, so could either go left, or it could go right. And so then there's going to be a sequence of these diamond structures, where they're constructed so that alternately either the Exist has a choice or the For All player has a choice, until you get all the way down to the bottom.

That is the graph we've built so far. Now, if we stopped here, then whoever ended up at this point, whether it was Exist or For All, would be the winner because there's nowhere for the opponent to go. Of course, that wouldn't be very interesting. So there's going to be more stuff we're going to add on, which are going to correspond to the clauses.

Now, how to think about what's happened so far? Maybe some of you might be guessing that when the Exist player over here has a choice, could have either gone left or right, that's going to correspond-- because this is supposed to be mimicking the formula game-- that's going to correspond to the Exist player in the formula picking the variable either true or false. So to the left or right is going to correspond to true or false.

And so let's just arbitrarily say left is going to correspond to picking it true and right is going to correspond to picking false. So the way I've set it so far is Exist ended up picking the first variable false, then For All picked the second variable false.

And the nth variable also got set false. Everything got set false so far. Who knows what happened over here, of course-- so just to understand what we've done so far.

Now I'm going to show you how to build the rest of the graph. So let's take away the green part. And now we're just back to the construction. The green part is actually how you use that construction.

So back to the construction here-- now, what do we want to achieve? By the time the players on the graph have got down to here, they've effectively made an assignment to the variables by going either left or right at each one of these diamonds. So the assignment is done.

From the perspective of the formula game, the game is over, and now one side or the other has won. Here, we want to build some extra structure here as a kind of an endgame, to make sure that the For All player gets stuck if the Exist player has made an assignment which satisfies this part of the formula. And the Exist player should get stuck if the For All player, if the assignment that they made does not satisfy the formula.

So let's see how we're going to achieve that with some additional structure. So there's going to be some extra node over here. Let me just tell you what the structure is, and then we'll argue why it works.

So going from this bottom node, we're going to start the second part of the graph. There's going to be a node which fans out to a node for each one of the clauses. And each one of those clauses, in turn, fans out to a node for each one of its literals.

So you see we have clause c1, and it has the literals x1, x2 bar, x3, x1, x2 bar, x3. So there's a node for each of the literals in clause c1, same for clause c2, and so on. Now we're almost done.

Now I'm going to tell you how to connect up these literal nodes. So x1, each node is going to correspond back to its own diamond. So now we're going to tie it back to the first part, where we made the assignment part of the graph.

x1 is going to connect back to the true part of the x1 diamond. And x2 because, it's negated, is going to tie back to the false part of the x2 diamond because it's an x2 variable. Similarly, x1 bar-- because x1 bar-- here's x1 bar in clause 2-- x1 bar is going to connect up now to the false side of the x1 diamond. And x2 bar is going to connect up to the false side of the x2 diamond, and so on.

I don't have the other diamonds here, but the x4 would connect up to the true side of the x4 diamond, for example. That's the whole construction. Let's understand why it works.

So the endgame here is we want Exist to win if the assignment satisfied all the clauses. And For All should win if there was some unsatisfied clauses. So why does this happen?

So let's put back an assignment here that was made. So the one I'm putting back, the assignment that they cooperatively built had x1 being true, x2 being false, and some other stuff. Now, why does this work?

So what we want to have happen now-- so now, the move proceeds over to here, and you need to arrange it so that Exist is the one who picks that node. If this would have been For All, just add an extra node to switch whose turn it is. But you want Exist to be up here. And what you want is for For All to be picking the clause node.

Now, here is the part to understand what's going on-- we want For All to win if this is not satisfied. So For All is going to make a claim. If it is not satisfied, there is some clause which is not satisfied. There's some clause which ended up being false in the assignment.

So the For All player going to say, "I think I won. And I won because clause number 1 is unsatisfied." So it's going to pick clause number one.

So here's the For All player's [INAUDIBLE]. Here's Exist player. And so the For All player picks the clause claimed unsatisfied. So over here, For All player says, clause c1 is unsatisfied. I'm going to move here.

Exist player says, "Uh-uh, I don't agree with you. The assignment, your line-- the assignment actually makes one of the literals true in clause c1. Let's say you made x1 was true, as in fact, it is here-- x1, in fact, is true.

So Exist is now going to pick the literal that was true in the assignment in that clause, that the For All player is claiming is unsatisfied. One of them is lying at this point. This is either now true in here, which means the For All player did not pick an unsatisfied clause, or the Exist player picked the false literal, which is the Exist player picked the false literal so the Exist player is lying.

Now, the moment of truth because let's see-- if Exist picked here, now it's the For All player's turn. It's connected back to the true side of the diamond. So that node was already used.

That's the only place that For All could possibly go. For All is stuck, and Exist has won the game, which is what you want because For All's claims are false and the Exist was correct in saying that x1 satisfied clause c1. So x1 is true, so there's nowhere for For All to go.

But compare that with the situation if the assignment, on this part of the graph, had gone through the false, had assigned x1 to be false, so the path had gone through the For All side of the diamond. Now this node would still be unoccupied, would still be available. So now the For All play-- so if Exist was claiming that x1 was true and it really was false, the For All player would be able to move onto that node.

And now it's the Exist player's turn, and the Exist would be stuck because this node down here is guaranteed to have been used. So that is the idea. It's really very beautiful, and actually not that complicated. You have to stare at it a bit.

So let's just see what happens, for example, if I had one other case-- maybe it's not necessary at this point, but if the For All player claimed it's the second clause which is unsatisfied in the mutually selected assignment, then-- so this is sort of the case where the For All player is correct, in that if the Exist player now says, well, I think x1 bar satisfied that second clause, but this assignment made x1 true, so x1 bar is false. And so the Exist player's lying this time.

And so now the For All player can take this last edge and go here, and has one last move to make. And then the Exist player's stuck. So the For All will win in that case.

So let's turn now, shifting gears entirely to a different part of space complexity. Instead of looking at polynomial space, which is very powerful, we're going to look at log space, which is comparatively speaking much, much weaker. So log space are the things that you can do when you only have enough space to write down, essentially, a pointer or some fixed number of pointers into the input. That's what you get when you have logarithmic space, because log space is enough to write down an index of something. So this is what you can do with a bunch of pointers.

And in order to make sense of this, we have to change the model of computation. Because if we use the ordinary one-tape Turing machine, just scanning across the input, just reading the input the way we've defined it, would cost space n. And so you can't even read the input if you have less than n space available, like log space.

So we're going to introduce a different model just to allow us to define this. And that's a model where it's going to have two tapes, where the part that contains the input doesn't count. And in order to make sure it's not being cheated, the input tape is going to be read only, but it doesn't count toward the space used. Only the work tape, which now can be written or read, is going to count toward the space bound.

So now what we're going to define is, using this definition for our space complexity, we'll define SPACE log n, the things that you can do if you have only a log amount of space here, on the work tape. So the length of the input is n. The length of the work tape is order log n. So we have the deterministic and nondeterministic associated classes-- SPACE log n and NSPACE log n. We're calling it L and NL-- log space and nondeterministic log space.

And as I said, that's just enough space to write down some pointers into the input. And let's do some quick examples. So if you take the language of palindromes, essentially, or ww reverse, that's in log space.

So here is a string. Here's a string that's in ww reverse. And the ordinary way you might use to test whether a string is of this form in ww reverse might be to cross off corresponding locations here.

But you can't do that. It's a read-only input tape. So you have to somehow avoid writing on the input tape, but still testing whether you're in this language.

It's not hard to do. You can use the work tape just to keep track of where your pointers would be. And in so doing, you can just make sure you're matching off corresponding locations with each other in the input. So log space, because of its ability to store some fixed number of pointers, gives you enough to test membership in this language. So this is in solvable in log space.

The path problem, which we've seen before-- you're given a graph, and a start node, and an ending node, or a start and a target. You're given a graph of an s and t-- it's a directed a graph-- and can I know, can I get from s to t? So that problem is solvable in nondeterministic log space because the way we would do that, not writing this down in any detail, but what you would do in your nondeterministic log space machine-- you have your input graph written here in the input, and you're just going to guess the sequence of nodes one by one, which takes you from the start node to the target node.

You're not going to write down that whole sequence in advance because that cost you way too much space to write down. The only space that you're going to use is to remember the current location where you're sitting. So you start out, you write down on the work tape the start node. Then you're going to nondeterministically pick one of the outgoing edges from the start node and look at its associated node, and replace that on your work tape, and keep repeating that. If you ever get to the node t, then you can accept.

And you also have to be a little careful-- I don't have this on the slide. You also have to make sure that if the graph has a loop in it, because that's not allowed in space complexity, so you're going to need also a counter to make sure that if you count up to-- you've gone through a number of nodes which exceeds the total number of nodes in the graph, then you can shut down that branch of the nondeterministic, because it's just going in a loop. If there's any path that connects s to t, there's certainly going to be a path that has the most of the number of nodes of the graph in it.

So path is in NL. This language here is in L. What's the relationship between L and NL? Well, certainly the deterministic class is contained in the nondeterministic class. That's all that's known.

Whether these two collapse down to be the same is unsolved. And so we're going to spend a little time next lecture exploring this. But let's first look at some of the basic facts about log space, sort of setting ourselves up for next lecture.

So first of all, anything that you can do in log space you can do in polynomial time-- in a sense kind of trivially, and in a sense we almost kind of really proved this already. Because if you remember, we said that anything that you can do in a certain amount of space you can do in time that's exponential in the amount of space, in the corresponding amount of space. So going from space to time blows you up exponentially. And the exponential of order log n is polynomial.

So the way we're going to prove this-- and again, we kind of proved this already but let's just go through it again, specific to log space. We'll say-- and this is going to be a useful definition for us anyway-- a configuration for M on w. So we also already talked about the configuration of the Turing machine M, which is just a snapshot, which is the tape, the state, and the head position.

When we have an input, which is sort of a read-only input, which is not being counted toward the space, we don't include that in the configuration. We just say it's a configuration of M on that particular input, but we're going to be counting configurations. And I don't want to count all the different inputs as well. I'm going to fix the input and count all of the configurations relative to that input.

And so the number of such configurations is just simply going to be the number of states times the number of head positions for the two heads now, times the number of possible tape contents, which is, as I mentioned, here is a number of different possible tape contents. It's d, which is the tape alphabet, to the order log n. And that's just going to be n to the k for some k. So it's going to be polynomial.

So that tells us that because the total number of configurations for M on w is polynomial, this machine can only be running for a polynomial number of steps. Otherwise, it'll be looping. It'll be repeating a configuration.

And so therefore, that machine has to run itself in polynomial time. So you don't really, in a sense, have to do any work. If a machine is deterministically deciding a language in log space, it's also deterministically deciding the language in polynomial time because that's all the time you can use when you have log space, unless you're looping, which is not allowed. So therefore, the same machine shows that you're also in P. I'll get to that in a second.

One thing I forgot to mention on the previous slide when I'm talking about the model is by the way, this model is not so unreasonable, where you have kind of-- imagine having a very large, read-only input and your local storage is much smaller. It's much too small to pull in the entire input. The way I used to explain this years ago was your input as a CD-ROM, which you guys probably barely even know what it is anymore, but you used to distribute software that way.

So the ROM was on a DVD or a CD, which contained whatever you're trying to-- like your software you're trying to distribute. It was some large thing, relatively, and so you imagine having a smaller amount of memory relative to that. So you didn't want to necessarily copy that whole thing into your storage.

Maybe even a better example now is you think of the input as being the entire internet. Obviously, unless you're Google, you can't download the whole internet into your own local memory, but you can have references, pointers into the input into different places. That's perhaps more analogous to this sort of read-only input Turing machine model that I'm describing.

And there's another fact that I want to mention, is that anything that you can do in nondeterministic log space you can do in deterministic space, but now with the squaring. And that's using the same proof. That's using Savitch's theorem, which you have to check also works down to log space-- same proof. So anything that you do nondeterministically in log n space you can use deterministically in log squared n space.

So let me just see if there's a couple of questions I want to answer here. The relationship between L and NL is not known to be strict. Nobody knows of an example. No one knows whether they're equal.

And have people looked at sublinear time classes? Yes, generally when you have nondeterministic or probabilistic-- which we haven't defined yet, but we will-- people have looked at those sublinear time classes as well. Deterministically, it doesn't make so much sense because then you can't even talk about the whole input.

OK, let me move on. This is my last slide. Let me see if we can do this before we break.

Not only is L contained within P, but the much stronger statement is that NL is contained within P. And for that you'll have to do some work because converting your nondeterministic log space machine to a deterministic machine-- obviously, you're going to have to change the machine. And so we'll introduce a new method here. Maybe we'll quickly go over this at the beginning of the next lecture. I don't like rushing through things at the end.

But for this, if I'm given a nondeterministic machine that runs in log space, I want to make a new machine that runs in polynomial time deterministically for the same language. And I'm going to define something called the "configuration graph" for M on an input w. And that's just you take M and w, its input, and you look at all of the configurations for M on w-- actually, configuration-- actually, that should be called the "computation graph."

That's what it's called in the book, but it's a typo. Yeah, I'll fix that next time. It doesn't matter. Computation graph, configuration graph, all the same.

Basically, you're going to take all of the configurations of M on w, of which we already observed are only polynomial in number. And they become the nodes of a graph. And the edges connect two configurations if one follows another according to the rules of the machine.

So here's a picture. This is some graph. The nodes of this graph are the configurations of M on w.

So each node here corresponds to a snapshot of the machine, a tape contents, head position, and state. So writing down all those different configurations, I connect one to the other, if I can get to C j from C i in one step legally on the machine. And then, the nondeterministic machine M accepts w exactly when there's a path from the start configuration to an accept configuration.

Let's assume we have just a single accept configuration, as we argued we can do one or two lectures back, if we clean up the tapes. So testing whether you can get from the start to the accept is the same as testing whether the machine accepts its input. And so because we can test whether there's a path in a graph connecting two nodes in polynomial time, we can solve this problem on this computation/configuration graph in polynomial time. And so we can figure out whether the nondeterministic machine accepts its input.

Sorry, that came a little faster than I like to do, so we'll see it again. So here's the polynomial time algorithm. You construct the graph, you accept if there's a path from the start to the accept, and you reject if there is not.

And so that tells us that not only is L contained within NL, but NL itself is also contained within P. So here's a kind of a nice hierarchy of languages. Not only do we not know whether L equals NL, we don't know whether L equals P. It's possible that anything you can do in polynomial time, you can do deterministically in log space, shocking as that might be, because this is a pretty weak class. But we don't know how to prove that there's anything different, anything in P that's not in here.

Last check in. So we showed that PATH is in NL. What's the best thing we can do about the deterministic space complexity of PATH? So deterministic. So this is nondeterministic log space.

What can we say deterministically about PATH? Hint-- this should not be hard if you think back to what we've shown very recently. Get your check in points. Closing up. Closing shop here. All set, 1, 2, 3.

Yeah, so the correct answer is log squared space, because this is just Savitch's theorem. We can do it in log space nondeterministically, so you can do it in log squared space deterministically. So this is what we did today. And as I mentioned, I will do this again on Tuesday's lecture, just to recap that.

All right, so I'll stick around a little bit for questions. And someone's asking me about the nomenclature. Why is it L and not L space? Because people don't usually talk about L time.

So L is sort of-- everybody knows the only reasonable option is space, so people just say L and NL. I mean, some of these names have a little bit evolved over time. And even now, some people talk about-- I call "time classes," some people call "detime classes." You can make different choices there.

Let's see. Good. Why does Savitch's theorem work for log n? You have to look back and see what you needed. And all you needed to be able to do was to write down the configurations.

And if you look back at how Savitch's film works, you're just needing to write down the configuration. So the deterministic machine can write down the configurations for the nondeterministic machine. They take exactly the same size.

And then you look at the recursion. And the depth of the recursion is going to be exponential in the size of the configurations. And so you're going to end up with a squaring again.

You have to go back and just rethink that proof. And you'll see nowhere did it need a linear amount of space. It works down to-- it actually does not work for less than log n. Log n is sort of the lower threshold there. And the reason for that is because you also need to store the input location, and that already takes log space.

The tape heads-- the tape heads already kind have kind of a log space aspect to them. And so if you're going to use less than log space, then funky things happen with storing the tape heads. And so less than log space usually turns out not to be interesting, very specific to Turing machines and not general models.

Yeah, so somebody's asking, suppose in the reduction to generalized geography from TQBF, if the formula had two Exists in a row, then you would do kind of the natural thing in the graph. Instead of having that spacer edge between the two diamonds, you could just have one diamond connecting directly to the other diamond without a spacer edge. And that would give you the effect of not switching whose turn it is.

Somebody is asking me just a general question, are people thinking about these open problems? I don't know. People don't say. There was a lot of work on problems related, that seemed to be related to those many open questions, like P versus NP, L versus NL or L versus P and so on, P versus PSPACE.

We'll talk a little bit more about some of that. There's some very interesting things that have been developed. I think there's a sense within the community that people are stuck, and you're going to need some sort of major new idea in order to push the thing forward.

So I don't know how many people are still thinking about them. I hope people are because I would like to see the answer at some point, or get the answer. I think about them sometimes myself, but one has to acknowledge chances of success are not high.

So we're a little after past the end of the hour here. Unless there's any other questions, if I didn't answer your question, I may have missed it, so you can ask again. But otherwise, I'm going to close this, close the session here. OK, bye-bye all. Have a good weekend. See you next week. Take care.