PROFESSOR: So welcome back to 6.890. Today we have the first of two guest lectures by Costis Daskalakis about PPAD completeness. Take it away, Costis.

CONSTANTINOS DASKALAKIS: OK. Thanks for the invitation to talk about PPAD. This is sort of an unusual class if you haven't seen it before. So part of the lecture is to describe the class and the motivation behind this definition. If I start with the definition itself, it may seem weird. So instead, I want to provide some motivation about the definition. And then sort of towards the end of the lecture, we're going to see the definition.

We're also going to see our first PPAD complete problem in the end of this lecture. And the next lecture, I'm going to describe more natural, if you want, problems about PPAD complete. OK. So the motivation for me looking at this class is some beautiful theorems in game theory, economics, and topology, namely Nash's theorem, Brouwer's theorem, and Sperner's theorem. And I want to describe those theorems before I get into the computational complexity of the resulting problems.

So let me start with Nash's theorem. So this table represents a game between two players. The rows correspond to the strategies available to the blue player. And the columns correspond to the strategies available to the column player. And then every box in this table writes down the playoffs that the two players get if they chose the corresponding strategies. In particular-- so this is a penalty shot game.

If the goalie dives left, and the kicker shoots right, then the goalie loses a point. The kicker gets a point. More generally, in a game, you have to specify the players n, the number of players n, the set of players n. Then for every player p, you have to specify the set of strategies available to this player. And also for every player p, you have to describe a payoff function Up that maps the strategies of everybody to the

reals.

So in particular, so Up of a collection of strategies for all the players corresponds to the payoff that player p gets if the players chooses these strategies from the corresponding sets. That's a general definition of a game. But in that picture, I have a two player game with two strategies per player for simplicity.

Now what game theory studies and tries to predict is what happens when two players interact in the way described by a payoff table and have developed various notions of equilibrium that may arise as a result of this interaction. The one that is of particular interest to us in this lecture is what is called the Nash equilibrium, which is a pair of randomized strategies for the two players of the game such that none of them has incentive to change their strategy given the strategy of the other player.

In this particular example, the Nash equilibrium, which is actually the unique one-- a Nash equilibrium which is the unique one of this game is for both players to uniformly randomly play each of their available strategies. And it's quite easy to check that this is a Nash equilibrium because given-- so for from the kicker's perspective, given that the goalie uniformly randomly dives left or right, the expected payoff from shooting left or right is exactly 0.

So in particular, any complex combination of these two strategies has an expected pay of 0. And he cannot improve it by changing the mixing weights in any way. So given what the goalie is doing, it's in the best interest of the kicker to play half, half. And vice versa. You can do the same calculation from the goalie's perspective. And given what the kicker is doing, the goalie cannot improve his expected payoff.

So more generally, in the general case, every player chooses a randomized strategy. A randomized strategy is a distribution over the strategy set of the players. So it's going to be-- with this symbol, I represent the collection of distributions over the set. And I can extend this function to distributions. So let's call this the extension of that function to distributions.

When I get as input a collection of mixed strategies, what I'm going to do is I'm

going to look at the expectation where every-- according to these strategies, this s1 drawn from x1, s2 drawn from x2 and so on and so forth, sn drawn from xn. I'm going to look at the expected utility under strategies, actions that are chosen according to this randomization. So that's the expected utility of player p if players choose these mixed strategies.

And Nash equilibrium in general has a definition, a collection, x1, x2, et cetera, of mixed strategies is a Nash equilibrium if and only if for every player p, the expected utility that p gets by playing this strategy is at least as good as the expected utility he would get if he switched to something else for any other mixed strategy he could possibly play. OK?

So if none of the other players change their strategies, there's no alternative mixed strategy for this player that may improve his payoff. So if that's simultaneously true for all players, then that collection is called the Nash equilibrium. And that example is just a special case of this definition. Any questions about what a Nash equilibrium is? OK.

Now there are games and games, right? And this game is a particularly simple one. First of all, it has two players and two strategies. But also it's very symmetric in some sense. If I look at every square of this table, the sum of payoffs in each square is 0. These types of games are called two player zero-sum games. And the existence of equilibrium in these games was shown by von Neumann. So von Neumann showed that any two players zero-sum game has such an equilibrium.

I mean, a priori, it's not clear if in a given game there is a collection of strategies that are a Nash equilibrium. The first one to establish such a result was von Neumann in the '20s, who showed that if a game has two players, and it's a zero-sum game, then there is always an equilibrium in that game. By the way, the equilibrium wasn't called Nash equilibrium at the time, because Nash wasn't around. It is known as the minmax equilibrium.

And in fact, later on when Dantzig was developing linear programming duality together with von Neumann, they realized that this theorem is actually implied by

the strong LP duality. They also noticed that using linear programming, you can compute equilibria in these types of games. In fact now-- but that took many years more. We know that this statement and strong LP duality are actually equivalent mathematical statements.

OK. So the existence of equilibrium in two player zero-sum games is equivalent to linear programming duality. All right? But how about general games? What happens if I go to this table, and I change it? Or I consider a game with more than two players? So what happens, for example, if I change this game in this way? Now I gave a big advantage to the goalie in the left, left choice of strategies.

Now before Nash was around, it was unclear if a general game has a Nash equilibrium. This is what Nash established in his celebrated paper. Nash showed that any game, any finite game has a mixed Nash equilibrium. So a Nash equilibrium and equilibrium in randomized strategies. Unfortunately, what is not known-- in this particular example, this is the Nash equilibrium. You can verify that this is actually a Nash equilibrium, what I'm showing there.

But the point of the slide is that there is a qualitative difference when you go from two players zero-sum games to general games. In 60 or so years of work after Nash's theorem, we have no proof using LP duality. And we have no polynomial time algorithms for finding equilibria in general games. So part of this lecture is to try to understand the computational complexity of this problem. Given a game, how hard is it to find a Nash equilibrium?

OK. So that's problem number one that I want to consider. Question?

AUDIENCE: So the thing that Nash proved, every game refers to any number of players and--

CONSTANTINOS DASKALAKIS: Right. Any finite number of players, any strategy set, any utility functions. No matter how it looks like, there's always a Nash equilibrium. OK? Other questions? This is, let's say, the actor number one in this lecture. So the second actor is Brouwer's theorem, which I want to illustrate in the next few slides. The statement of the theorem is very simple. It says that given a function that maps a convex and

compact subset of the Euclidean space to itself-- so I don't know.

In this setting, compact means basically closed and bounded. So if you have a function that maps a convex, closed, and bounded subset of Euclidean space to itself, it must have a point that doesn't move under the function. And I want to illustrate this with a couple of examples. And for my examples, I'm going to choose for D the disk, the unit disk.

I'm going to note that all conditions are needed actually for the theorem to be true. If you remove any of these things, closeness, convexity, or bondedness, the theorem fails. So I'm going to do a bunch of examples, and you can think about what may happen if you remove one of these conditions. So my first example is to map the disk to itself by just rotating the disk. Now there is an obvious fixed point in this map, which is the center, as you may imagine.

There's actually no other fixed point. What if I take this disk, and I shrink it, and then I shift it somewhere inside it? I don't know. Maybe it takes a little thinking, but you should believe me that there must be a fixed point somewhere there. Or a fixed point still exists if I kind of crumble the disk and move it. As long as I don't tear it apart, that's a continuous map. And there's still a fixed point. And you probably colloquially have heard that if I take a map of Cambridge, and I throw it on the ground, then there is a point on the map that sits on top of the corresponding point that it represents.

This is a special version of this theorem because effectively when I'm throwing a Cambridge map to the ground, I define a mapping from real points to points in the map. And by this theorem, there is a point that doesn't move. OK? So that's what Brouwer's theorem is saying. And the proof of Nash actually uses this theorem. To show that the Nash equilibrium exists, what Nash did is he used Brouwer's theorem. And the next slide, I want to illustrate the way Nash gets to prove the Nash equilibrium's existence by using Brouwer's theorem.

And I'm going to restrict myself to the simple game we saw in the beginning, the penalty shot game. And as I said, for this specific game, you don't need Brouwer's

theorem to prove the existence of equilibrium, because this is a two player zero-sum game, and you can even do it with linear programming duality. But it's a simple enough example that will help illustrate the proof of Nash's theorem.

So here's what Nash does. He starts with a game. And then he defines a continuous function, in this case, from the square to itself, from the unit square to itself in a way that the fixed points of that function are actually exactly Nash equilibria. The way the function looks, where basically, what is the square? The square represents the probability-- the horizontal line is the probability by which the kicker shoots right. The vertical axis is the probability by which the goalie dives right.

And then the function is basically tracking which of the two players has incentive to change his strategy at every point in this quarter. So in particular, let's consider this square over here. In this square, both players play left. And the kicker is actually unhappy. If both players play left, the kicker wants to increase the probability by which he is shooting right. So this is what this arrow represents. So you want to increase the probability over here.

Similarly, if you're here, they play opposite actions. Hence, the goalie wants to change and so on and so forth. What Nash is-- I'm not going to write down Nash's function, but intuitively, this is what it tracks. It tracks at every combination of mixed strategies, which players have incentive to change their actions, their randomizations and in which direction?

This is what Nash's function does. And in fact, to help you a little bit, I'm going to even color the unit square, depending on the direction of the deviation. So this is the picture that I get. It's very easy to locate the fixed point as the point where all colors meet. And this is exactly half, half, which is exactly the Nash equilibrium over here. This is a schematic proof by picture. So this is an illustration of Nash's proof.

Any questions about Brouwer's theorem or the proof? OK. The last actor-- so the last actor is Sperner's lemma. It's a lemma in combinatorics, but it's very related to topological statements. So the lemma considers-- there are multiple versions of the lemma. And I'm going to show it in two dimensions on the square. There are multi-

dimensional versions of the lemma. But I'm going to restrict myself to the plane.

And the lemma is going to start with a triangulation of the grid. And I'm going to color this triangulation with three colors. More generally, if I'm in n dimensions, I'm going to be coloring things with n plus 1 colors. Here I'm in two dimensions. I'm coloring with three colors. So the way I'm going to color this square is the following. First, I'm obliged to use this coloring on the boundary. Notice that this boundary has the property that there is no red color here. There is no blue color here. There is no yellow color here on that edge of the grid.

But otherwise, I'm allowing you to color the internal vertices in whatever fashion you want. All I want is to respect the boundary coloring. So Sperner's lemma says that no matter how you color the inside of the square, there is always a tri-chromatic triangle. In fact, it says something more. It says that there is always an odd number of tri-chromatic triangles. Can you see one in this picture?

AUDIENCE: I hope so.

CONSTANTINOS DASKALAKIS: You would hope so, right? So there are five in this case. Here they are. And later in the-- I mean, I'm actually going to show the proof of Sperner's lemma because it's instructive for my purposes later in this lecture. It's really remarkable. I mean, a priori, I mean maybe you believe there's a tri-chromatic one. But it's not at all. Obviously, there should be an odd number. And the proof is actually going to easily give also the odd number as well. So that's the lemma. That's Sperner's lemma. Any questions?

AUDIENCE: Can you repeat where the legal coloring is?

CONSTANTINOS DASKALAKIS: Yeah. So the legal coloring says that either fix the color to be like this. Or more generally, as long as you don't use red here, don't use yellow here, and not use blue here, you're good.

AUDIENCE: I don't understand the general statement.

CONSTANTINOS The general definition is that-- OK. So we have this square. There must be-- OK. So

DASKALAKIS: more general, no yellow in these vertices. There should be no blue on these vertices. And there should be no red here. That's the general statement. As long as you respect that, then no matter what you do inside, there is a tri-chromatic triangle.

Or you could just fix the coloring to be that. It doesn't really matter. In fact, if you have such a coloring, respecting these three properties, you can enlarge the square by adding an extra layer that has exactly the coloring I'm showing here. And you can notice that if your original coloring respects these properties, if I add this specific coloring in this extra layer, I'm not going to be creating any triangles in this layer. So in fact, I can reduce one instance to the other.

All right. So this is the statement of Sperner's lemma. There is a multi-dimensional version in n dimensions. I'm using n plus 1 colors. But I'm not going to state it here. Now again, I'm going to hand wave to actually show you how Brouwer's theorem is actually implied by Sperner's lemma. So in fact, one proof of Brouwer is actually going through Sperner. Here's the high level proof of the argument.

Given a function-- I'm going to do it on the square, on the unit square in two dimensions. So suppose I'm given a continuous function from the square to itself, what I'm going to do is I'm going to prove for all epsilons-- for all epsilons, I'm going to show that there exists an approximate fixed point with approximation epsilon. using Sperner's lemma. That's what I'm going to do with Sperner's lemma.

And then I'm going to use a compactness argument to get this down to 0. So I'm going to create in fact a sequence of approximate fixed points with smaller and smaller epsilons. And I'm going to use compactness-- remember, this is a compact set-- to argue that there is an exact fixed point. So Sperner's lemma can be used for this first part of the proof. And roughly speaking, the way it's used is the following.

What you do is you first triangulate the square. And the diameter of your triangles, which is very rough in this picture-- the diameter of this will depend on the constant of absolute continuity of your function. So if the function is ellipsis, it's just going to depend on epsilon and the ellipsis constant. So depending on the ellipsisness of the function, there's going to be a finer triangulation.

Now having triangulated the square and adding this extra layer outside, you're going to color the vertices that are inside depending on the direction of the displacements f of x minus x, using exactly the same coloring schema I showed you earlier for the illustration of how Nash showed the existence of equilibria using Brouwer. So I'm going to color those internal vertices using this coloring scheme, depending on the directions. And then for the extra layer that I added on the outside, I will just use my standard coloring.

Now what's cool with the way I chose my colors here-- I mean, this is my standard coloring, except I have permuted the names of colors. Sorry about that. I think, you know, red took the role of blue and stuff like that. But it doesn't really matter. What is cool about the way I chose my colors is that if a point is here, I know that is not going to be using-- essentially, it can only be in the boundary between red and yellow. But if you're here, and the displacement is yellow, you're going to be going outside.

That's a bad thing. So the way I chose my colors, I know that some property like this is going to happen. Some of the edges of this cube are going to be missing some of the colors. For example, here, there is no red, because if there were red, then I would have to be going down outside my square. And my function is mapping the square to itself. So that can't happen.

Similarly, over here I cannot have a yellow and so on and so forth. So that's going to be a valid instance of Sperner's lemma. And by Sperner, there is a tri-chromatic triangle in this scenario. Here it is. You could see the fixed point at the back. That's a coincidence actually because I'm going to do the compactness after. But I'm going to do Sperner's lemma. Now a tri-chromatic triangle is actually going to-- one of the corners of the tri-chromatic triangle will have this property if you choose the diameter of your triangles to be-- it's correctly chosen.

So that's, roughly speaking, how the proof works. Yeah?

AUDIENCE: The fact that Sperner's lemma shows that there's an odd number of such triangles

means there's an odd number of [INAUDIBLE] not carry over?

CONSTANTINOS DASKALAKIS: This carries over, except in degenerate games where it doesn't hold. But in a non-degenerate case, it will hold, not by direct use of Sperner's. Because Sperner is always going to go through approximations. So there is another way to prove the existence of Nash equilibria in two player games using what is called the Lemke-Howson algorithm. and this Lemke-Howson algorithm has a similar flavor as Sperner. And it also has an odd number of solutions. And this is still true. So for two player games, it is still true that there is an odd number of solutions if the game is non-degenerate. I'm not sure what's the status of multiplayer games, because then there are irrational solutions. So combinatorics give you approximations but not exact fixed points. Yeah?

So yeah. So the fact that there is an odd number of solutions calls for this combinatorial problem, Sperner. But after you do compactness, then I don't know what happens. So all bets are off. But for the specific case of two player games, there is an alternative proof that carries over. OK? Cool. OK. So these are the three problems that I'm interested in.

And now it's the time where I'm going to get into a bit of complex theory discussion. And in fact, this complex theory discussion will lead to the fact that NP, the standard concept that we use for understanding complexity, is not sufficient to address these problems. And that will motivate PPAD. But first of all, let me give intuition about why Sperner hard. And here's the situation I want to be looking at.

So I want to be looking at the grid of size 2 to the n. So in particular, I'm not going to be drawing the grid explicitly for you. And I'm not going to be listing the colors of the internal vertices as a list. So what I want to do is I want to create a succinct description of a coloring of an exponentially large instance in the following way. So I'm going to be imagining the existence of this grid with its boundary colored in this specific way.

And what I'm going to give you is I'm going to give you a circuit that takes as input the coordinates of points inside this square and outputs a color. Now so this circuit

only decides the colors of the vertices inside. And I fix the colors of those vertices to be what is shown in the picture. Thus, that's a succinct description of a coloring of 2 to the 2n vertices. And this coloring satisfies the conditions of Sperner's lemma.

In particular, there is a tri-chromatic triangle. And what the problem is asking you to do is to find such a triangle. Describing this triangle can be done with polynomials complexity because all you need to do is to output the coordinates of the three points participating in the triangle. So the output description-- I'm not asking you to do something crazy, right? So the output can be described in polynomial size. The input-- and what's problematic with this problem is that the size of the graph we're working with is exponential in the input size.

So this circuit has n input bits. So this graph is in principle exponentially large in the description of the problem. So in particular, I cannot just go and check every triangle about whether it is tri-chromatic or not. I have to do something smarter to solve this problem. And in fact, we have no other-- essentially, no smart solutions to this problem. OK. So that Sperner's lemma. That's a way to define the computational version of Sperner's lemma.

Similarly, you can define Nash and Brouwer. I'm only going to define Nash. Brouwer is a bit more complicated. So in the Nash problem, I'm giving a game which is described by the number of players, the strategy sets, which I enumerate, and the utility functions of all players. This is the product of all the sets. And I'm also giving you an approximation requirement epsilon. And I'm going to comment on that in a second. What I want you to do is to find an epsilon Nash equilibrium of this game.

Now what's an epsilon Nash equilibrium of the game? An epsilon Nash equilibrium is a collection of mixed strategies such that no player has more than an epsilon incentive to change his strategy. So in this definition here, I would add a minus epsilon here to make it an epsilon Nash equilibrium. So I have not more than an epsilon incentive to change my mixed strategy. That's what an epsilon Nash equilibrium is. Now why did I have to define the problem in this way?

There's a specific complex theoretic reason why I'm defining my problem as an

epsilon Nash instead of an exact Nash equilibrium. What is that reason? Yeah

AUDIENCE: Maybe the exact Nash equilibrium makes [INAUDIBLE].

CONSTANTINOS DASKALAKIS: Right. So Nash's proof is via Brouwer's theorem, which is a topological statement. And there's no guarantee that the output-- if I were to omit this epsilon from this definition, there's no guarantee that the output has polynomial size complexity. Maybe a Nash equilibrium is comprised of irrational numbers, which I cannot specify. Maybe it's not even algebraic. Who knows? But it is algebraic.

So approximations are in fact-- Nash, his paper in '51, not the original one but the one that appeared a year later. He actually provides an example of a three player game that only has irrational equilibria. So I'm working with epsilons to avoid this issue. Coming back to two player games, you can actually show that any two player game has a rational equilibrium whose probabilities are of polynomial complexity in the description of the game.

So in this specific case of two player games, you can actually omit the epsilon if you want from the description of this problem. But for general games, you add a third to make this question computationally meaningful. Yeah?

AUDIENCE: Is it always the case that there's an equilibrium that's a root of the polynomial bounded complexity in the description of the problem?

CONSTANTINOS DASKALAKIS: I believe this is true, yeah.

AUDIENCE: Can you do the same thing?

CONSTANTINOS DASKALAKIS: You could, yeah. You could in principle do that. I'm not going to define Brouwer. But you can imagine in Brouwer similar things. You have to take care of similar things and define epsilon approximate fixed points, et cetera. Now let's look at NP. The variant of NP that I'm interested in is that of search problems in NP, so function NP.

I don't want to bore you, but I'm going to go through the definitions. So a search problem is defined by a relation of inputs and solutions such that a pair x and y is in

this language defined by this problem if y is a solution to x. So I expect you to know all this. And I'm going to go fast through these definitions. But what's important here is this definition. When do we say that the problem is total?

We say the problem is total if for every input there is a y that is a solution to the problem. For example, Sperner's problem is total because for any instance of Sperner's problem, there is actually a solution. For any input of this form, you know by Sperner's lemma that there is a solution. So that problem is a total problem. Similarly, for every input to Nash, there is a solution. So these problems are total problems. Let's see. OK.

Now a search problem is an FNP, function of NP, if there is a polynomial time algorithm and a polynomial function so that two conditions are satisfied. First, if the algorithm on input x and y is 1, then and only then y is a solution to x. And also for every input to the problem, if there is a solution to that instance, then there is one whose size is polynomial in the input of the problem. So you have solutions that are polynomially large.

And so this is FNP. And TFNP is the class of problems that contains all problems in NP that are total. And based on the previous discussion, all the problems we're interested in, Sperner, Nash, and Brouwer, are TFNP. Sorry. TFNP. Both in FNP but also in TFNP. OK. And I guess let me remind you about FNP completeness. A problem in FNP associated with an algorithm and a polynomial function is poly-time reducible to another one associated with an algorithm and a different bound.

If there exist two functions, f and g, one of them is mapping inputs to l to inputs to l prime. And also, you have two properties. And it's important to look at these properties carefully because they're related to a point that I want to make. So for the reduction to be valid, you also need this property, that for every x and y, if the algorithm of problem l prime on the transformed input and y says 1, then you can take that solution and transform it into a solution to the instance to l. This is what it says.

Well, this one says that if the transform instance has no solution, then the original

instance also shouldn't have any solution. And a search problem is FNP complete if it's both in FNP, and every problem in FNP is polynomial time reducible to it. And of course, finding a satisfying assignment is FNP complete. But now we reached the point that I want to make. What I want to claim is that I cannot possibly reduce SAT to any of the problems that I'm interested in.

In particular, I cannot show that there's no Karp reduction from SAT to any of these problems. And the reason is this property. Recall, this property was saying that for every input to problem I, if the y' is a solution to the transformed instance, then I can convert that solution to a solution to the original instance. But now if I is SAT, and this is Sperner, then no matter how you transform the SAT instance to a Sperner instance, there is going to be a triangle that is tri-chromatic, setting this to 1.

So I should be able to then convert this triangle to a satisfying assignment. But what if those instances are unsatisfiable? That possibly cannot hold if your two problems are SAT and Sperner. So there's no reduction. There's no Karp reduction from SAT to Sperner. So you can't show any of these problems for that matter. So you cannot possibly argue that these problems are FNP complete. Another thing you may try is to get Turing reductions from SAT to these problems. But that has other issues. It basically would imply that NP equals co-NP.

So there's no way basically. These types of problems don't-- if you want type check, they don't type check. These are problems that are total, and these are problems who's complexity really resides with the fact that it's hard to distinguish satisfiable and non-satisfiable instances. So LP completeness is not the right language, if you want. It's not the right framework to understand the complexity of these problems.

And the exercise through which we're going today-- and I think it's instructive-- is how to go about characterizing problems that don't fit into traditional complexity theoretic frameworks. So in particular, what I want to understand is how to go about defining a complexity theory of total search problems inside FNP. Clearly, these problems actually are inside FNP. These are FNP problems. They're TFNP problems. So how do you go about coming up with a complexity theory of total

search problems?

And the way you go about it is to look at the proof of existence of solutions in these problems and understand the combinatorial underpinnings of the existence argument. So if you want the 100 feet overview of the methodology that I'm going to follow, it's this. First, I want to identify the combinatorial argument of existence responsible for making all these problems total problems. And then I want to define a complexity class inspired by this argument of existence.

And the litmus test of my approach is if the resulting complexity class actually tidily characterizes the complexity of these problems. So what I'm going to do is I'm going to show you the proof for Sperner's lemma, and that's going to inspire the definition of PPAD. So this is what I'm going to do in the next few slides. So here's the statement of the lemma. If your boundary is legally colored, then there is always an odd number of tri-chromatic triangles, in particular at least one.

Let me try to prove it for you. The first thing I'm going to do says I'm going to introduce an artificial vertex at the bottom left of the square. Here it is. And I'm going to color it blue. Now no matter how you color the internal vertices of this square, because I have fixed by coloring on the boundary to be the one you see here in this picture, I know that by adding a blue vertex over here, I'm creating an artificial tri-chromatic triangle.

I'm going to call the triangle the beginning of my walk. And now I'm going to basically define a walk in this grid. Consider this is a big factory. The factory is partitioned into triangular rooms. And I'm starting my walk in this factory from this vertex here, from this triangle here. Now my walk is going to be doing the following thing. It's going to be going from triangle to neighbouring . Triangle and the transition rule is the following.

I'm going to look in the room I'm in, and I'm going to try to find a red-yellow door. If I find such a door, I want to try to cross it having red on my left hand. That's the rule of my walk. Red-yellow doors. I want to cross them with red on my left. That's important. I meant to say yellow on your left. Sorry about that. So let's go back to

this picture. I created these artificial triangle over here. I knew it's going to be tri-chromatic because it only considers vertices on the boundary.

And I have fixed the coloring of the boundary to be this specific one. I know it's going to have a red-yellow door which I can cross having yellow on my left. So let me cross that door. Now I'm entering into a new room. In this particular case, there's nothing interesting about this room because this still touches the boundary. So no matter how you color the internal vertices, you couldn't affect the coloring of this boundary.

So it's also going to have a red-yellow door, which I can cross having yellow on my left. And the interesting business starts now. I entered into a new room, and depending on your coloring, maybe it's already tri-chromatic. So how do you color this into blue? This would already be a tri-chromatic triangle. why? Because you entered that room through a red-yellow door, and you encountered a blue color.

What is cool about it though is that if it's not blue, then there's going to be for sure another door which you can cross having yellow on your left. In this particular scenario, this was a yellow. It wasn't a blue. So this was not a tri-chromatic triangle. But because this was a yellow, I enter a room through a red-yellow door, and I find a yellow. There is for sure another red-yellow door I can cross having yellow on my left.

So this is what I'm going to do. And I'm going to keep doing this until-- what can go wrong, right? So the claim that I want to make is that if I keep doing this procedure, I cannot exit the factory. For me to exit the factory, there has to be a door on the boundary which I can cross having yellow on my left. Now because-- and that's where the coloring of the boundary becomes important. Because I have fixed the color of the boundary to be the one you see in this picture, you can verify that there is no red-yellow door you can cross to go outside.

The only red-yellow door that exists is this one. But if you're inside, you cannot cross it going to the outside having yellow on your left. And that's where having left on your left hand side is important. This is the only red-yellow door. But you cannot

cross it going outside. You can only cross it going inside. So you cannot exit the factory. So your walk is going to keep going. And there are two possibilities now. So one is that it's going to keep going. And then it's going to enter into a new room through a red-yellow door, and it's going to find a blue.

In that case, my lemma is proven, not the odd number, but at least I've proven to you there is at least one triangle because I entered into a room. I found a blue. I entered through a red-yellow door. That's a proof of existence. But there is one thing that can go wrong, though. What is that? I can loop around. So what I can do is I can-- so this is my factory. I started over here. I start wandering around this factory.

And then I sort of got into a room that I had already visited before. And then I get into a loop, and I go forever. That's the problem that may happen. I claimed over there that this cannot happen. Why can it not happen?

AUDIENCE: Coloring.

CONSTANTINOS DASKALAKIS: Yeah, let's try to color it. Let's zoom into here. So this is a room that I got into at the beginning through a red-yellow door having yellow on my left. So originally, I got into this room like this having yellow on my left. Then I exited this room through a red-yellow door having yellow my left. So this door, potentially it was a red here. So I exited this way. But then I cannot possibly enter through this door, because that's a red-red door. Or this could be a yellow, in which case, I exited from here. But still now I cannot get back from that door, because that's a yellow-yellow door.

So this cannot happen. So my walk cannot get back into an already visited room, and it cannot exit the factory. So the only thing it can do is just stop because there's a finite number of rooms in this factory. But the only way it can stop is if it finds a blue when it enters a new room. So that's the proof of existence. If you follow my walk, indeed, you're going to get into a tri-chromatic triangle.

And this way, I finished the first part of the claim. Any questions about that? Yeah.

AUDIENCE: The same argument give you the odd number though?

CONSTANTINOS DASKALAKIS: It does, yeah. What is it?

AUDIENCE: If you find another one, then it has to [INAUDIBLE].

CONSTANTINOS DASKALAKIS: Right. Let's try to do that. So let's start from a different triangle. Let's start from here. But if I start from here, there's no red-yellow door I can follow. So that doesn't give me anything. But let's start with somewhere where there are red-yellow doors. So let's start with here. Now I can follow the same kind of business. I can get into doors. I can cross red-yellow doors having yellow on my left. In this case, that got me into a tri-chromatic triangle.

Now let me go back here and actually do the word backwards, so have it going backwards, having yellow on my left. So this is what happens. So I found a tri-chromatic triangle. And these actually came in pairs. The fact that this gave me a tri-chromatic triangle must mean that this has to give you another one because by the same token, I cannot sort of get into a loop while going back. On the other hand, there are some triangles where that would give you-- so you get another pair over here.

These don't give you anything, because if you start here, you don't move. But here you can get loops. If you start from the inside, you can get into a loop. The point is that all triangles come in pairs, except for this walk, which started with an artificial one. So all tri-chromatic triangles come in pairs, except for the tri-chromatic triangle that I found in my original walk. Thereby, there is an odd number of those.

So it's kind of-- its pretty cool, right? Let me actually try to abstract this proof a little bit and sort of focus more onto the core argument why this happens. So here's what I did, really. So this is my space of triangles. This is my set of triangles. I want to define a graph where the vertices are the triangles. And the adjacency of triangles-- so two triangles are connected if they share a door, and you can go from one triangle into the other one through this door having yellow on your left.

So for this reason, all vertices have out degree and in degree at most 1 because

every triangle-- no matter how it's colored-- has at most one red-yellow door which you can cross having yellow on your left to go out. And it has at most one door which you can cross having yellow on your left going in. So just the minute definition of my transition rule, which was cross a red-yellow door having yellow on your left, defined a graph on the set of triangles that has in degree and out degree at most 1.

Now a graph with in degree and out degree at most 1 is a very simple type of graph. It looks like this. It's a bunch of isolated vertices, a bunch of directed cycles, and a bunch of directed paths. And in fact, the way I define my position rule, you could check that all degree 1 nodes must be tri-chromatic. If you're tri-chromatic, there is either one door you can exit, one red-yellow door you can exit having yellow on your left. Or depending on how the coloring is, one door you can use to enter the triangle having yellow on your left.

So only tri-chromatic triangles can have total degree 1. Isolated vertices are those who don't have red-yellow doors. And degree 2 are those who have two doors, one with which you can exit having yellow on your left and one with which you can enter having yellow on your left. So what I want to summarize is that the mere definition of my transition rule defines this graph in a way that all these unbalanced vertices are solutions to the Sperner problem. All the vertices were solutions to the Sperner problem.

And in fact, one solution was artificial because it was that triangle that I created by adding the extra node outside. So all degree 1 vertices are real solutions, except one of them. And by the structure of the graph then, there has to be an odd number of solutions. This is implied. So to summarize, the mere description of my transition rule defines this graph in a way that all unbalanced vertices are solutions. And because these have to come in pairs by parity, there is an even number of solutions.

Except one was artificial. So hence, there is an odd number of solutions. So in particular, what I want to argue is that all these very beautiful, very interesting theorems that I showed in the beginning, Nash, Brouwer, and Sperner-- the real

argument of existence, besides sort of mathematical work, mathematical obfuscation, if you want, is this trivial statement in combinatorics. If you have a directed graph with some unbalanced vertex, then there must be another unbalanced vertex.

You obviously know this theorem. The proof of this theorem is pretty clear. If you have a directed graph where some node has a different number of outgoing and ingoing edges, then there has to be another such node. And effectively, what I did over here is I described a directed graph, and I identified this artificial vertex, which was unbalanced. It only had an ongoing edge but no incoming edge. And by parity, I knew there must exist another one.

So in fact, I didn't even have to use my walk argument and stuff. All I had to claim is that the mere definition of my transition rule defined this directed graph. And this directed graph had some unbalanced vertex. And I know that somewhere in this graph, there has to be another unbalanced vertex. But all I'm saying is that the core of it is this trivial lemma.

And now your question will be, well, what the heck? This lemma is trivial. So we took so many algorithms, Karger's class and so on and so forth. We've worked with graphs. We know how to identify these things, right? Given a directed graph, it's trivial to find an unbalanced vertex. But the point is that you can sometimes define exponentially large graphs succinctly. And you can ask people to find your unbalanced vertices in exponentially large graphs.

So that's what PPAD is about. In fact, if I give you a Sperner instance by providing you a circuit, we'd only be working with exponentially many triangles in the description of the circuit. So that's why the problem is difficult. The problem is difficult because the guy we're working with is exponentially large in the description of the problem, of the instance. So the PPAD class is trying to capture this structure.

And we're right about the point where I'm going to define the class formally. So you ready for this? Oops. OK. The PPAD class, defined by Papadimitriou in the '90s-- actually, this is the general version of the paper. The PPAD class is giving you an

exponentially large graph. The vertex set of the graph is 0, 1 to the n. The graph is specified in a weird way. I'm going to give you two circuits that get an n input bit, and n bit input and have an n bit output.

And using these two circuits, I define an edge between a string and another string if a weird condition is satisfied. So now we directed that from v1 to v2 if this condition is satisfied. Let me try to parse these symbols there. But before that, let me call this circuit the possible previous circuit. And let me call this circuit the possible next circuit. Now this circuit takes as input string, and it outputs a candidate neighbor of that string.

And similarly, this-- sorry, a candidate father of the string. And takes as input a string and outputs a possible child of the string. And I only established an edge, a directed edge from string v1 to string v2 if string v2 believes that its father is v1, and v1 believes that its child is v2. If they agree on their sort of parenthood relationship, I add this edge. OK? Now you may ask me, what the heck? Why didn't you just define this graph in a different way? Why did you have to have this possible child and possible father relationship and describe the graph in this way?

And the answer to this question is that no matter-- so the answer to this question is, no matter what circuit you give me, that always defines your graph with a very specific property that we're going to see in the next slide. Back to the definition. So I'm going to explain the intuition behind this. But back to the definition, describe an exponentially large graph by giving these two circuits. And I establish the edges between nodes only if this condition is true.

And what I'll ask you to do is the following. Given these two circuits, if the all 0 string is unbalanced and gives an exponentially large graph that I defined, then I'll ask you to find another unbalanced node, which by parity exists. If the 0 string is unbalanced in this exponentially large graph, then by parity argument, there must be another unbalanced node. I'm asking you to find it. That's the definition of this end of the line problem. And PPAD is the class of all problems that are in FNP that are poly-time reducible to this problem.

Now this sounds weird. So let me show you how the graph defined by these two circuits looks like. And this will be hopefully familiar. So I'm copying here the condition for putting an edge between two strings. The set of things I'm looking at is 0, 1 to the n. This is the condition for when I put an edge from string v1 to string v2. Now notice that by definition, every string has at most one possible child. And every string has at most one possible father.

So the in degrees and out degrees of this graph are at most 1. So I'm implicitly describing a graph that looks like this. And what I ask you to do is to check the following. I ask you to check if 0, the 0 string, is unbalanced, meaning it either has a father but not a child. Or it has a child but not a father. If it's balanced, you don't need to do any work. But if it's unbalanced, which is easy to check locally-- you just query the candidate next child and check if that child believes that 0 is its father or vice versa.

If 0 to the n is unbalanced, then by parity, because the graph looks like this, there must be some other unbalanced vertex, at least one. And I'm asking you to find me any of these unbalanced vertices. So in particular, you don't need to find the unbalanced vertex that's in the other end of the path that starts at 0 to the n or finishes at 0 to the n. You can give me any of the other unbalanced vertices if there are any. But by parity, I know that there is at least one unbalanced vertex.

So PPAD is this problem. I describe an exponentially large graph by giving you two circuits. Now no matter what circuits I gave you, I know the graph looks like this. Now in this graph, if 0 to the n is unbalanced-- if a 0 string is unbalanced, you know that there must be another unbalanced vertex. Any of them is good as a solution to the problem. Yeah?

AUDIENCE: So how then do we know that 0 to the n doesn't point to a previous other string that points to 0 to the n? How do we know that 0 to the n--

CONSTANTINOS DASKALAKIS: So in that-- no, it can happen. It could be that this is 0 to the n. It has a child that believes that this is its father. And also, v believes that 0 to the n is its child. And 0 to the n believes that this is the father. This creates a 0 to the n which is balanced. In

that case, you don't need to do any work. So the statement of the problem is asking you to do work only if 0 to the n is unbalanced.

So end of the line tells you if 0 to the n is unbalanced, which you can trivially check locally-- you don't need to look at the whole graph to decide whether 0 to the n is balanced or not. So to decide whether 0 to the n is balanced, you need four queries.

AUDIENCE: Couldn't you have an exponentially long segment?

CONSTANTINOS DASKALAKIS: Yeah, but the point is-- that's the whole point. So if I query-- if v--

AUDIENCE: I see.

CONSTANTINOS DASKALAKIS: --is the possible parent of 0 to the n, and u is the possible child of 0 to the n, then the state of the world could be this. It could be this. Now if also this guy believes that 0 to the n is his father, I'm going to make this edge solid. That's the fourth query I'm going to make to my circuits. If this guy believes that 0 to the n is his father, I'm going to also make this edge solid.

If both are solid, then 0 to the n is balanced. I don't need to do any work. If this is missing, then 0 to the n unbalanced. I need to do work and so on and so forth. OK? But with four queries to my circuits-- so in polynomial time in particular in the description of my input-- I can decide whether 0 to the n is balanced or not. And if it's unbalanced, then I need to do a lot of work. And in fact, we have no better than exponential bound for how long it will take me to find the other unbalanced node.

AUDIENCE: Can you make all such graphs of this type with small circuits?

CONSTANTINOS DASKALAKIS: Mm-mm. I mean, no. I mean, you're asking, if I pick an arbitrary exponential graph, if it's always compressible into polynomial size? I mean, it depends on how phrase the question. So if you allow me to get a huge p-- p is part of the input. p is a part of the input. So now if p is a lookup table, that's a trivial instance, actually. So the instances that are hard are those where p is a succinct thing. It's polynomially in n size but still defines an exponentially large graph.

If p is a huge thing, a lookup table that keeps track of all connectivities of all vertices with each other, we're getting into algorithms territory, where I give you an adjacency list for every node. So this is a purely algorithmic problem that we can trivially solve. So if I get into the regime where my p, my circuit description, is polynomial in the size of the graph, that's an easy instance.

But if you're asking the question, can I always compress an exponentially large graph into two circuits, p and n, that are polynomial in size, then that's not the case. OK. So PPAD-- notice that this end of the line problem is in FNP because I can just guess the unbalanced vertex, different than the 0 string. And I can check easily if it's also unbalanced with four queries. So it is in FNP.

And I hope you believe me that Sperner isn't PPAD, because just the proof of Sperner was exactly the same kind of thing. So I succinctly described this graph that had exactly the same structure. So really, what inspired the definition of this class is the proof of Sperner's lemma. Now I sort of argued that you can prove Brouwer via Sperner. I gave this hand wave-y, high level picture showing that. And also Nash through Brouwer.

So in fact, PPAD definitely contains the problems that I'm interested in. That was the litmus test. The litmus test is I went through this exercise to define this class PPAD based on the existence proof of Sperner's lemma. And I successful got all these interesting problems inside PPAD. But the litmus test is-- the success of my exercise is whether I get also tight reductions. So if I also get that these problems are as hard as any other problem in PPAD, now I can call my thing successful.

So in particular, as an example, the definition of NP is that SAT is NP complete. So NP was defined with SAT in mind. But it also captured exactly the complexity of SAT. So that was a successful definition of a complexity class. So the litmus test for this exercise that I went through with you is whether actually those problems that I'm interested in are also PPAD complete. So what we showed with Goldberg and Papadimitriou a bunch of-- I guess 10 years ago almost-- is that actually the opposite reductions are also true, that all these problems are computationally

equivalent.

In fact, I didn't show that these are equivalent to PPAD. These were already known. These results were already known since Papadimitriou's original paper, that PPAD exactly is equivalent to Sperner and Brouwer. What we show together is that Nash also is equivalent, is in the same league of problems, basically establishing that all of these problems are computationally equivalent. Finding a Nash equilibrium is a computational equivalent to Sperner's lemma, computationally equivalent to any problem in PPAD and so on and so forth.

And the high level of the reduction-- I'm going to hand wave about it-- is that we started this generic PPAD instance. And what we do is we envision embedding this instance into some geometry. Because this a combinatorial problem, our goal eventually is to reduce it to Brouwer and to Nash, which are continuous problems. So the first thing we did is we embed this graph into a cube in a way that's sort of like every path or cycle corresponds to a path that moves around this cube without intersecting itself.

So this is the first thing we do. Then we find a way to solve the resulting embedded PPAD instance, reduce it to a Sperner's instance. And then in fact, the core of the proof-- and I'm going to actually talk about this more-- is to reduce this problem into a problem that asks you to evaluate an arithmetic circuit. Now after you get that PPAD reduces to this evaluation problem, it's really-- I call it approximate arithmetic circuit SAT. So I'm going to get into what this means.

But it's really a satisfiability problem for arithmetic circuits. But after you get PPAD to reduce to this problem, it's easy to get to Nash and all the other implications, all the other known PPAD completeness reductions that people have established. So what I want to describe is what this problem is about. It's the equivalent of circuit SAT, which is NP complete. This is called arithmetic circuit SATs, and I'm going to show you what it means. Question?

AUDIENCE: So in the course of this reduction, do we get to replay our game for Nash, like does this three carry through to the--

CONSTANTINOS DASKALAKIS: Yes. So you get actually two players. You can get two player Nash. Yeah. So it establishes that all games are-- if they're not two player zero-sum, they are PPAD complete. So the input to the problem is a circuit. The circuit comprises two kinds of nodes, viable nodes and gates. So gates are represented with solid circles. Viable nodes are with-- I don't know-- empty circles.

So the viable nodes are x1 through xn. The gate's nodes are g1 through gm. Now the gates can-- I can choose gates from the following set of possible gates. I'm going to explain what these gates are meant to do. But this is an assignment gate. This is a plus addition gate. This is a subtraction gate. This is set equal to a constant gate. This is times a constant gate, and this is a comparison gate. The subtraction and comparison gates are non-symmetric in their inputs. So they have a special input 1 and a special input 2, similarly for the comparison gate.

And I also have directed edges that connect variables to gates and vice versa. But there are no edges between variables. So there are no edges between the gates. And also, the loops are allowed. That's important. So you can have loops. For example, in this picture, you have a loop between vertices. And actually, there are no inputs to the circuit. In circuit SAT, you want to set the input so that the output is 1. And you don't have loops. You have a DAG. And your gates are just binary gates.

Here we're in a different situation. We don't have inputs. Our gates can have loops. And they're not binary. They're arithmetic gates. The goal is to assign 0, 1 values to the variables so that the gate constraints are satisfied. So now what are the gate constraints? So the output of an assignment gate should equal the input to the assignment gate. If we have an addition gate, the output of the addition gate should be the sum of the inputs, except everything should line 0, 1.

So it's the minimum between the sum of inputs and 1.

The subtraction gate is similar. So the output to the subtraction gate is the difference of the inputs. But you can't go below 0. The set equal gate is-- set equal to a constant gate is sets y to ba. But it doesn't allow it to leave 0, 1. And then

multiply by constant gate multiplies the input with alpha, with a. But also it doesn't allow it to go below 0 or above 1. And these are pretty straightforward.

What is slightly weird is the comparison gate, which has a weird behavior and actually must have this behavior for the problem to be PPAD complete. You can show that. The comparison gate has to be a 1. The output has to be a 1 if the first input is bigger than the second input. It has to be a 0 if this first input is smaller. But if they're equal, it can be anything. It's allowed to be anything. Actually, if you think about it as a continuous gate, it cannot just jump from 0 to 1 without being able to take all the intermediate values. OK?

It can be anything. Any value is allowed if the inputs are equal. So that's the arithmetic circuit SAT problem that is PPAD complete. First of all, this problem is total. No matter what circuit you write down, I can prove to you that there is always an evaluation that works. That's not a priori obvious. It's not a priori obvious that no matter what circuit I describe using these gates has a satisfying assignment. But I claim this is true.

I'm not going to show it to you. It is true though. And it must be true because this problem belongs in this family of PPAD complete problems. So it has to also inherit all the properties that this class of problems shares. Now let's try to work out the details of what the computation is. OK. So I want to argue that in this-- I want to claim that the only satisfying assignment is that everything is equal to 1/2 actually. And let's try to verify it.

So suppose that a was not-- actually, no. I take that back. So a must be 1/2, and b can be-- no, actually they're all 1/2. Sorry, sorry. They're all 1/2. I don't take it back. So let me try to argue that. So suppose that a was bigger than 1/2. So c must be 1/2, right? That's by the condition of the gate. This is set equal to 1/2. So this must be 1/2. That's obvious.

Now I claim that a cannot be bigger than 1/2. If a was bigger than 1/2, then this is bigger than this input of the comparison gate. So the comparison gate would output a 0. But this set equal operation would have to set this guy to be 0. So 0 can't be

bigger than 1/2. Let's do the other direction. Suppose a was smaller than 1/2. If a was smaller than 1/2, then the comparison gate would [INAUDIBLE] 1. This would set it to 1. So how can 1 be smaller than 1/2?

So the only possibility is that it's actually 1/2. And if it's 1/2 and 1/2, then it's setting 1/2. Because the inputs are equal to the comparison gate, you can set this to be anything. So everything being 1/2 satisfies all the conditions of the gates. So that's the problem we're trying to solve. make The claim is that this problem is PPAD complete. And this problem is going to be the starting point for all our reduction next time, in particular showing that Nash's theorem is PPAD complete.

I want to add the last point. After we published our paper, Chen, Deng, and Teng actually improved these results to fuzzy gates. So they-- actually, we allowed in our original paper exponential noise in the outputs of the gate. So this is an easier problem now. So I give you a circuit. And I want you to find me an assignment that satisfies all gates approximately. So now in the original paper, we could accommodate exponential noise in the size of the input and the fuzziness of the gates.

These guys showed that you can even accommodate any polynomial noise, inverse polynomial noise that you want. And it's still PPAD complete. That's an easier problem than the exact problem obviously. But in fact, they're computationally equivalent because they're both PPAD complete. So now next time, what I'm going to show is I'm going to use this problem to show that Nash and market equilibrium are PPAD complete. I'm going to show that some problems in combinatorics, such as fractional hypergraph matching and Scarf's lemma are PPAD complete.

And then if we have time, I'm going to show you other existence arguments that give rise to different interesting complexity classes.

AUDIENCE: Named after Papadimitriou?

CONSTANTINOS DASKALAKIS: Yeah. Many people ask that. So it means polynomial parity argument of directed graphs. That's a good question. Yeah. I should have said that. OK. Thanks a lot.

[APPLAUSE]