PROFESSOR: All right. Let's get started. So we're continuing on the theme of linkages.

And just a brief recollection from last time-- and also changing definitions a little bit and throwing away some details we won't need-- we have the idea of a graph, which is just vertices and edges. And then if we added edge lengths, if we put a number on each of the edges, we've got what we call the linkage. And then if we actually embed that picture into the plane, or into 3D, or whatever space we're working in, into d-dimensional space, then we call that a configuration of the linkage.

And in general, there might be many configurations of one linkage. If it's going to fold or move around, there is infinitely many configurations of a linkage. And what today is about is basically when there's only one configuration, versus when there's many, at least locally.

So this is the concept of rigidity. So if we have some linkage configuration-- some configuration of some linkage-- then we call it flexible if it can move from that configuration in a non-trivial way. So we want a non-trivial motion starting from that configuration.

And you'll notice that in my definition of linkage here, I did not specify that some of the vertices could be pinned to the plane or pinned to 3D, because we needed that a lot last time. A bunch of our gadgets had fixed vertices-- pinned vertices. I'm going to throw that away, and the price I pay is that whatever linkage I build can float around in my space.

And so I need to add this non-trivial specification to say that it's not just a rigid motion. It's not just a translation and/or rotation. That's rigid motion. So you ignore translations and rotations, for example, by pinning.

But you can also just see-- what are all the motions I can do from here? If it's just in 2D, if it's just the three dimensions of I can translate in two ways, and I can rotate in one way, then we don't call it flexible, we call it rigid.

And today is all about distinguishing rigid scenarios from flexible scenarios. And yesterday-- or last class-- was all about making things flexible, and making things flexible in an interesting way so that we computed some polynomial. Today, we're really interested in the case where we can make things rigid.

There are some places, some situations, where you want things to not move, like you're building a building. Unless you're really cool and you're making a reconfigurable building, you want it to not move. You want it to be rigid. And so some classic examples of this are the truss.

This is a standard planar truss of some nearby bridge. I have another truss for the architects. It's an octet truss-- or at least various sections of it in the Brussels airport. You see these a lot in airports, especially in ceilings. And trusses all over the place, especially in bridges.

And they are rigid linkages. You've got vertices, you've got edges. In principle, they could hinge there, though they're not usually designed to allow hinging. But the point is even if they are allowed to hinge, it will be a rigid 3D linkage configuration.

And these are very simple examples we would like to generalize to understand the general picture. Turns out, the general picture in 3D is not really well understood. It's quite complicated. But in 2D, which is what we'll be working in today, there's a very good answer, and it's very clean, and you can characterize all the good rigid 2D structures to some extent.

Let me elaborate a little bit. One annoying thing about rigidity is that it depends on the configuration, not just the linkage or the graph. So let me draw you a picture.

Suppose we have a rigified square. I'm going to work in two dimensions. Add some triangles. So this thing is rigid. A bunch of triangles in the plane. And let's say I do that, verses-- and I'm going to allow crossings here. It doesn't really matter too much.

If I draw this right-- and I've done a little better. It's easier when I have a grid-- my graph paper. In this situation, the configuration should be rigid, because this part is

rigid, and these three bars are held taut, because these vertices are effectively pinned to the rest of this framework. And this is at full length. You can't really hinge here.

Whereas if I flip this triangle through that vertical line, flip that over, I get this triangle. And I did the same thing for the other triangle. I didn't have to. But now, suddenly-- and you can't do that by a motion. But if I consider this other configuration where that's the case, then suddenly these guys, which have exactly the same lengths as they did before, have some slack, and now you can think of this as a quadrilateral which can flex separately.

So this guy's flexible, and this guy's rigid. This is annoying, because we'd like to-- think of a truss, or something like this. You don't think about exactly how it's drawn. You just think, oh, well, there's this structure. It's got vertices. It's like squares with diagonals. Does it work if they're skewed squares? This presumably works.

We think, oh, triangles are rigid. It shouldn't matter exactly how they're drawn. And it turns out, most of the time, it doesn't matter how they're drawn. This is kind of a very special case, and I want to formalize that notion of "special case" and "most of the time."

Because in particular, if I gave you a particular linkage configuration, and I ask you, is it rigid? As I mentioned in the last class, this is coNP-hard. So almost certainly there's no good algorithm to tell you whether a given linkage configuration like this-- versus this-- is rigid. That's disconcerting, because I want to have a good characterization of rigidity. And this is true, even in two dimensions.

So we're going to change the problem a little bit so suddenly it becomes easy. We're going to be a little bit less precise. We're not going be able to distinguish between this and this, but almost all the time-- like if this thing just had a little bit of slack, it would be flexible. So we're going to ignore the fact that occasionally this thing is taut, and most of the time, this picture would be flexible most of the ways you draw it.

All right. This is the idea of generic rigidity. We've talked about generic situations before in a single vertex piece patterns. We said, well, generically, no two angles are ever going to be the same if we just sort of perturb everything a little bit. We're going to take the same idea here. And so, for example, no two edge lengths are going to be the same, but we're going to assume a lot more than that to make life easier.

The general goal is to think about the graph-- not the linkage, and not the configuration. I want to say that rigidity is usually a property of the graph. Most the time, it doesn't matter what your edge lengths are, it doesn't matter exactly how you draw it, it just matters how things are connected. Are they triangles? What is that combinatorial structure? The geometry falls out of it. I need one term so I can talk about graphs in a realization.

This is just shorthand. It'll make my life a lot easier. It's a configuration of a linkage of the graph. We have three levels. The graph, which is just the combinatorial infrastructure, linkage, where you add the edge lengths, and then a configuration where you draw it. I'm going to jump all the way to the end and say, if I start with a graph, I want to go all the way to a configuration. That's a realization.

And so you can think of a realization as just some mapping. It looks exactly like a configuration. Just forget about the edge lengths. You can put the vertices wherever you want, because you haven't said what the edge lengths are supposed to be.

So by taking a graph and drawing it in the plane-- this is going to be our situation-- you can compute what the edge lengths become, and then see, does it flex given those edge lengths? So as soon as you draw in the plane, the edge lengths become fixed. But before that, they're free to be whatever you want. You just embed every vertex into the plane, and you get a realization.

And now I want to define the notion of a generic realization. And this is a subtle notion, and it's a little tricky to define. So I'm going to give the intuition. And I will give a definition-- we'll see another definition next class, in fact. But it's a pretty intuitive notion, especially the way I'll be using it.

I want to somehow avoid things that are rare-- degenerate situations. And mathematically, that means lower dimensional situations. In general, a realization-- remember, we can think of it as a point in d times n dimensional space. If n is a number vertices, d is the dimension of space that they live in. Then we could just write down the coordinates of every vertex separately. And the realization space is just that. You can do whatever you want.

So we would like to throw away lower dimensional subspaces of that that just make our life easier. So for example, I'd like to say no three points are co-linear. So if I take three points in the plane, it's pretty unlikely they lie on a common line. It happens with probability zero if I had random points. Even if I didn't have random points, I just take arbitrary points and then perturb them a tiny bit in a random neighborhood-- little epsilon disk-- then with probability zero, they will be aligned.

So I can say a whole bunch of things like this, maybe no four points con-cyclic, which means lies in a common circle. You know, Latin. In 3D, you'd assume no four points are coplanar. All these sorts of things. In general, what we can throw away is any non-trivial rational algebraic-- how many adjectives can I get in here? I think that's all of them-- equation.

You can, for example, write the fact that three points lie in a common line as a polynomial with integer coefficients-- or definitely rational coefficients. And generally, you take any polynomial you want, it explains some geometric property. If it has rational coefficients, and it's nontrivial, meaning that it's not always true-- so sometimes it's true, sometimes it's false. That's all I mean here-- then you can just-- suppose that doesn't happen.

Because if you have an equation, that describes a lower dimensional space. You start with d times n dimensions. If you add a constraint on the coordinates of some vertices-- over here we have all the coordinates of all the vertices. That's what we're thinking of. You take any polynomial over those. If that holds, then that was degenerate, and you can throw that away.

And it may sound weird, because there are infinitely many of these equations, but for the mathematicians, they're only countably many. So if you take these spaces, you take a whole bunch of lower dimensional spaces times countable number, it's still lower dimensional. And so you can afford to throw away all this stuff.

If that doesn't make sense, don't worry about it. I will tell you various times when we're just assuming that we're generic. It'll seem like magic, because whenever I get into a situation I don't like, I'll just say, oh, but that's not generic. And we move on. But this is the formal version that makes that safe. We'll see another version that's a little more intuitive next time. But we need other concepts.

So this it implies that degenerate situations are lower dimensional. And what other good facts do we have? So lower dimensional means this is going to happen with probability zero, if you perturb your vertices a little bit.

So in particular, this scenario-- this is sort of the generic picture, where there's some slack here. This is extremely rare, because in particular, we have four points co-linear. And that's not going to happen in a generic situation. So we would say that this graph-- a bunch of triangles, plus these three bars connecting the endpoints-- is generically flexible. Most of the time, it will be flexible.

And fun fact is that for any graph-- and this is not obvious, but it's true-- it's either generically rigid or generically flexible. Generically rigid means that all generic realizations are rigid. And generically flexible means all generic realizations are flexible. I'm going to start abbreviating, because these adjectives get really long, especially with the adverb form.

So when I said rigidity is usually a property of the graph, this is what I meant. Either you take all the generic realizations of your graph, and they're either all rigid or all flexible. There's going to be the lower dimensional subsets where maybe it's the other way around. I'll give you some examples of that.

So here-- what the heck did I draw? That's funny. One of the edges moved on me. If I take a picture like this-- OK, this is flexible, because I have a quadrilateral here

that can flex independent of the triangles. But suppose I add a bar between this vertex and this vertex which somehow connects the two sides of that quadrilateral. When this special case, the way I drew, where these triangles are identical and this is a nice rectangle, this thing is flexible.

You take this triangle, and I want to flex the quad, so if I rotate the whole thing like this-- I think it works. It twists at the same time. Yeah, it's a little hard to see. But because these guys are all parallel, this thing can keep them all parallel, I think, and keep all the distances the same.

AUDIENCE: All of them are parallel and the same length?

PROFESSOR: All parallel and the same length. It's necessary for this to work. But you take a generic drawing-- if I perturb this at all, it will be rigid. so this is rarely flexible, but generically it's rigid.

And I have an example of the reverse. The way I've drawn this, where all these guys in extension meet up, and all these guys are parallel, and so on, you can check-- it's a little hard to see-- but it's rarely rigid. In this case it's rigid, but generically, it's flexible.

I wouldn't take this on faith. For me to check this, I had to constructed in Cinderella, and then see that it moves. And when these things are almost parallel, it moves just a little bit. And you can see that in the limit, when all these things line up nicely, it doesn't move at all. The point is that there are these exceptions. That's all you need to believe.

But most the time, we're going to get the right answer. And so, the rest of this class is about characterizing when a graph is generically rigid or generically flexible. It's a nice problem, because it just depends on the combinatorial structure, but occasionally, it will give the wrong answer. If you've set things up with very special geometry, it might be the other way around. Question?

AUDIENCE: Is it fair to think the rigidity is happening right before it no longer works?

PROFESSOR: Yeah. Right. The rigidity is kind of happening right before it breaks. In some sense, there is a nontrivial motion there, it just happens to be super short. It's hard to define that formally, but intuitively, that's what's happening. And you can think of this thing as being a little bit wobbly, but not technically flexible. There's actually a formal notion of wobbly in the rigidity literature, which we probably will talk about next class. Would this be wobbly? Probably. I'd have to think about it. Wobbly is a computable notion. It's nice.

So rigidity theory is actually pretty old. In the mechanical, structural engineering worlds, it goes back to Cremona In the 17th, 18th century. The mathematics even is pretty old. The next big theorem we'll talk about is from 1911. It's probably one of the earliest formal mathematical treatments, and it started to address exactly this issue. In two dimensions, which graphs are rigid, which are flexible in the generic sense? It's become pretty popular and interesting in the last 10, 20 years, and a lot more action-- a lot more theorems and whatnot.

Today we're going to talk mostly about old things. Next class we'll do some newer things in the rigidity world. I got interested in rigidity because it turned out to be really important for folding linkages. Not just telling whether a linkage is rigid, which is important for things like building rigid structures, but for actually folding things and showing that you can fold the linkage from any configuration to any other.

Turns out rigidity is really useful for that. And that's not at all obvious. It's kind of a surprise, and a fun surprise. We'll be talking about that next class. Today, sort of classic stuff-- understanding when 2D graphs are rigid. I'll give you an idea why 3D is much harder.

OK, we need one more notion. This is a real long-winded thing-- minimally generically rigid graph. So the new addition here is minimally. This is going to be some generically rigid graph, and it also has the property that removing any edge makes the graph generically flexible. So it's minimal in the sense that I can't get rid of any edges and still be generically rigid.

So the idea is, let's characterize minimally generically rigid graphs. If I can

characterize those, then the generically rigid ones are just those plus some more edges. You could throw in extra edges-- it only makes things more rigid. It's more constraints. But I'm really interested in the boundary between generically region and generically flexible, and this is exactly the boundary on the rigid side-- just barely rigid.

So first thing-- if you've done structural engineering, this will start to look familiar-- is how many edges should a minimally generically rigid graph have? And we can think of this in a very intuitive way, which is-- let's see. If I don't have any edges, every vertex can float around separately.

So the number of degrees of freedom-- the dimensionality of my realization space, in fact-- is in two dimensions two times m. I'm going to do to 2D first. Every vertex has two degrees of freedom. Now I would like to reduce the number of degrees of freedom to what number?

AUDIENCE: Zero.

PROFESSOR: Zero. It's often a good answer, but sadly not the right one here.

AUDIENCE: Two.

PROFESSOR: Two. Close.

AUDIENCE: Three.

PROFESSOR: Three. Right guess. We'll just keep trying all the integers.

I mentioned this at the very beginning that if the best we could hope for is that there are three degrees of freedom, we'll never be able to get rid of the two translation degrees and the one rotation degree in two dimensions. In general, it is d times d plus 1 over 2 is the number of dimensions of rigid motions in d dimensions.

And here we start with dn. So that's in d dimensions. This is really how many edges you should have, because if I have exactly 2n minus 3 edges, I will have started with 2n degrees of freedom. Every edge that I add is one equation, so it should reduce

my dimension by one in the generic sense-- it's a generic case-- and the best I can hope for is to get down to three. If you put in more edges, you won't get any more rigid. I mean, you'll still be rigid, but if I want the minimal case, this is really the best I should hope for.

And this is just a rough sketch. This is proved more formally in the notes here, but this really is a formal thing.

Again, if you take some polynomial equation, most of the time it's going to define a space that is one dimension lower. So in the generic situation, every equation you add will remove exactly one degree of freedom.

There are rare scenarios where it doesn't remove any, and they're rare scenarios where it removes more than one, and I have some pictures of them here. But most of the time, this is right. And we are in the generic case, so most the time is all we need. So this gives us an idea.

And about how many people have heard of 2n minus 3 before in this structural engineering context? A few. And sometimes, people end the story there. But this is not enough to be generically rigid. This is a necessary condition, but it's not sufficient. And you can check all these examples. It should match up. I haven't checked them myself.

All right. Let's start characterizing this thing. We're going to see two characterizations of minimally generically rigid graphs. First one is very intuitive, nice to work with by hand, but doesn't-- at least as far as I can tell, it doesn't turn into an algorithm. I think we've tried in past years, but it has not. And the other one is more algorithmic, but a little bit hard to intuit.

So this is what both theorems look like. A graph is minimally generically rigid in 2D if and only if-- for Henneberg, this is from the 1911 result-- it can be built.

Oh, by the way. Maybe the reason people haven't seen 2n minus three is because they're used to the 3D case. There it's 2n minus 6. How many people have seen 3n minus 6? Same people. All right. Thought I'd try. All right. You guys should take

structural engineering.

So this is the form of the theorem. The idea is we're starting with a single edge connecting two vertices, which is minimally generically rigid. All it can do is rotate and translate. And the claim is whatever you want to build, I can build it using just two types of operations.

First type is I take something that I've already built. I add one new vertex, and I attach it to two existing vertices. So everything in the circle is old, and the new thing is that vertex and those two guys.

AUDIENCE: Everything is old and rigid.

PROFESSOR: Yeah. This is something that you could already build from this-- I mean, in particular, that will be minimally generically rigid. But that's not what we're saying right now. You start from the edge, and you just keep doing this. So I can turn it into a triangle, for example. Just keep taking things that I already know how to build, add one vertex and two edges.

The other type of operation, I take something I've already built. I take three of the vertices in there, and there's already an edge between them-- at least one. There might be more. And I add three edges connecting to a new vertex, and I delete that old edge. So I add three, I remove one.

Either I add two and remove zero, or I add three and remove one. This should make sense, because if we're going to get about two edges per vertex, this is just the starting case. If I want two edges per vertex, I really should only be adding two edges for every vertex that I add. So these are two things you could do.

And you could imagine a more complicated ones where I add four edges, remove two. You don't have to worry about that. You only have to worry about these two scenarios. That's enough to build everything.

Maybe we can do an example before we prove this theorem. I'm not going to prove all the parts. I'm going to give a sketch. In particular, to prove it, I need technology

that we'll develop next class. So it's a little hard to do in a self-contained way. But let's start with an easy example.

You may remember this one. When I drew everything parallel and the triangles were identical, then this thing was flexible. But most of the time, generically, this is rigid, and we can prove that by doing these operations. So somehow I start with an edge. I have to do operations to end up here. That's a little hard to-- yeah. Question.

AUDIENCE: Does the removed edge have to be between two of the three vertices?

PROFESSOR: The removed edge has to be among those three vertices. And there might be more than one. You would just remove one of them. You get to remove whichever one you want. Good question.

So if we were lucky and adventurous, we could just try going here and hope that we end up there. But we don't want to draw all the possible things we could make, because that's all minimally generically rigid graphs. It's a lot easier to think backwards, as often the case in puzzles.

If I want to make this, what was the last operation that I did? It must have been one of these two. And every vertex here has degree three, has three incident edges. So probably it was a Type II operation. So there's going to be a Type II operation that results in this thing.

Let's say, I don't know, this vertex. And here I still have to be a little bit lucky, but it's not quite as lucky. OK so I'm going to try not to cheat here. So we remove that vertex, and we also remove some edge among these guys. So I'm going to guess that I removed that edge. I think that'll work. I have to be careful. I could have removed this edge. There are two cases. I'll explain later what we need to check here.

Well, now I see a nice Type I operation. I'll just get rid of that vertex. Good. So Type I operation. I'm left with this, and I see two more Type I operations. I'll do them one at a time. Ta da. I think that's right.

We can play it forwards just to double check, but I started with an edge. I added one vertex connected to the existing two. I added another vertex connecting to two existing. I added another vertex connected to two existing. And then the tricky one-- I added this vertex here, I connected it to these three, and I removed this edge. So because this works forwards, this theorem tells us that graph is minimally generically rigid. Now we have a way to test.

Now, the algorithm is a little tricky, because I had a choice here. In this case, it didn't matter. In the notes, there's an example where if you do the wrong choice-- you go backwards and you add in the wrong edge-- it doesn't work. You don't become minimally generically rigid, essentially because one part gets too many edges, and another part has too few. Here, I got it balanced, and it worked out.

And when you're good at visualizing these, this is an easy way-- like on your problem set two-- this is an easy way to check, to prove to me that things are minimally generically rigid. Just have to be a little careful. If you fail, and you fail to get back to start in this way, that doesn't tell you much. You just may have messed up. You may have made the wrong choice along the way. But when it works, you know that it works, because you can play it forwards.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. By forward, I mean starting from the single edge and following the arrows in the way that they're pointing. So going backwards is a little bit of a lucky black art, whatever. But once you succeed, you can play it forwards and say, oh yeah, that clearly works. Another question.

AUDIENCE: Do you know if it matters whether you do Type I or Type II operations first, if you have choices?

PROFESSOR: Does it matter Type I versus Type II? I don't think it matters in general, just with Type II, you have to be careful. Type one, I like to do immediately, because there's no choice. You just do them. Type Is are always safe. Type II, it depends which edge. And I should call these anti-type II operations when you go backwards. Anti-

Type II, it depends which edge you add. So I like to do Type Is whenever I have the chance.

All right. Let me sketch a proof of this theorem. There are two directions we need to prove. We need to prove that if we build such a thing, it is minimally generically rigid. I'm going to start with that. And the other one we need to show is that everything can be built in this way. The first direction is not too hard, although it's still not trivial. And this direction works in any dimension, in fact. It doesn't need to be two dimensional. It's the other direction that's harder.

So let's suppose I have a generically rigid graph, and I do a Type I operation. And I get a graph call G prime. I claim that the resulting graph will also be generically rigid and vice versa. So doing a Type I operation does not affect generic rigidity. This may be obvious, but let me explain it to you.

There's again two directions you need to prove. And let's start with-- which one do I start with?-- this way. So I want to show that if the new graph is rigid, than the old graph is rigid. An easier way to think about that is if the old graph is flexible, then the new graph should be flexible. Hm. That sounds all right.

So I have G flexible generically. I want to show that G prime is flexible. So here's G-- what I called old before-- and it flexes somehow. And now I have this guy that I add on. These bars are rigid, but it can come along for the ride. If I move these two points somewhat, I'll still be able to draw this guy.

It is defined by the intersection of two circles, one circle centered here, another circle centered here. Those two circles actually intersect in two points. But if I move these guys continuously, this guy can track continuously along the intersection of those two circles, except in one special case, which is when the three points are collinear. So if this is the old, and here's my new guy, and this is taut, and these guys move away from each other, then this guy fails to exist. Boom.

But that's not generic. So here's where I get these genericity. This in particular would have three points collinear. So that's forbidden. So in a generic situation--

and I should be putting generically here-- if I can generically flex the original graph, I'll still be able to generically flex after I add the vertex. Cool. That's pretty easy. The other direction-- I mean, it's almost the same.

So now if I have G rigid, I claim that G prime is generically rigid-- I should be putting generic everywhere here-- for essentially the same reason. If I can't move these vertices-- this vertex is again defined by the intersection of two circles. It could be here or here, but we're only thinking about continuous motions.

Motion means continuous. I can't instantaneously jump over here. I have to stay right there. These guys don't move-- and I mean that in a relative sense. Of course, they can translate and rotate-- but if they don't move away from each other, this guy can't do anything interesting. So if the original thing is rigid, the new thing is rigid. I'll start going a little faster, but that's the idea. This is a start, but what we really care about is minimal generic rigidity.

And this is also if and only if. If your original graph is minimally generically rigid, then your new graph from a Type I operation will also be minimally generically rigid, and vice versa. So minimality is also preserved. So this is a super safe operation. I won't prove this.

But it's almost the same proof, except instead of thinking about G, you think about G minus 1 edge, because that's what you worry about in the minimal situation. We've already shown that the generic rigidity part is preserved in either direction, doing Type I or anti-Type I. But for minimality right here, we want to think about removing any edge, and whether that gives generic flexibility.

So we just do the same thing, but with G minus e, and G prime minus e, where e is somewhere in the old part. And it's the same argument. And so you'd write substitute all instances of G with G minus e. That's pretty old and geeky. Probably no one knows what that means, but you get the idea. All right.

Now, we get to Type II operations. It's a little more subtle. if I said this with Type II, it would be false. So life it's harder. And this is all about the fact that anti-Type II

operations are not unique, and you've got to be careful in how you do it. It does work in one direction. If the input is minimally generically rigid, then the new one will be.

But it doesn't work in the other direction. Now at this point, I'm not going to prove this. This is actually a lot harder to prove. At least, I don't know of a simple, straightforward proof like this one, where you just construct it. The easy way I know how to prove it uses next class. So we're not going to bother proving it. It should be intuitive enough. Certainly in thinking about degrees of freedom it's correct, but you've got to be more careful than that.

I want to talk more about the reverse direction. At this point, by the way, we have proved one direction of our big theorem. We've proved that if you can build something with these two operations, it is minimally generically rigid, because when we start out with an edge, this is the base case of our induction. This guy is minimally generically rigid.

And then every operation I do, I know I'm safe, because I start with G being minimally generically rigid. If I do a Type I operation, it will still be minimally generically rigid, and same thing for Type II operations. So in that direction, we're golden.

But I'd really like to know, is there some converse of three, this third property? And I will tell you a converse of the third property. Suppose I have a minimally generically rigid graph G prime. And I'm not assuming that it's made from a Type II operation yet. But suppose it has a degree-3 vertex-- vertex with three incident edges. Then G prime is the result of a Type II operation of some minimally generically rigid graph G. So this is saying that there is hope in working backwards.

Here I had two choices of Type II operations. I think in this case, it didn't actually matter, but in general, it matters which one you choose. And this is saying there is a choice, there is an anti-Type II operation, a backwards II operation, that results in something that is minimally generically rigid. And if it's minimally generically rigid, by induction, I can keep going on. And so eventually I will get back to start. And this is

what we need to show in order to believe one half of this theorem that if your thing is minimally generically rigid, there is a way to build it like this.

And the way you build it is if there's a Type I operation-- anti-Type I operation do that. If there's an anti-Type II operation, do the right one. Let the right one in. Do the right instance of an anti-Type II operation that gives you something that's minimally generically rigid. If you preserve minimal generic rigidity, you know you can keep going, because you have a smaller graph And so by induction you can keep going.

I'm not going to prove four either, because it again uses technology we haven't yet developed. But there's one more part of this theorem that we haven't proved. This sounds great. We do anti-type I operations. When can we do those? We can do those when we have a vertex of degree-2. Then we can get rid of it. And what property four over there says that if I have a vertex of degree-3, I can get rid of it, and add some edge to compensate. But what if I don't have any vertices of degree-2 or 3? That would suck.

So this is now the other half of this theorem. All right. Well, we know by the thing I just erased that the number of edges is 2n minus 3. So just to recall, we're assuming now we have something that's minimally generically rigid. We already did a degree of freedom analysis to show the number edges must be 2n minus 3.

Now, we need to somehow find either an anti-Type I operation or an anti-Type II operation so that we can go all the way to a single edge and find a Henneberg construction. Any suggestions? People know graph theory? All right. On the one hand, we have the number of edges, and the other hand, we care about the degrees of vertices. Anyone know a relation between those two things? Yeah.

AUDIENCE: Two. Every edge contributes two to the total Henneberg [INAUDIBLE].

PROFESSOR: If I take the sum of the degrees of the vertices-- sum over all vertices-- this is really intuitive. I add up how many edges are incident to this vertex, then I add up how many edges are incident to this vertex, then I add up how many edges are incident to this vertex. I count every edge twice, because it has two ends. So some of the

degrees is twice the number of edges. I'll write that this way. E is the set of edges that's the size of that.

This is called the handshaking lemma. It says if everyone in this room shakes hands with everyone else-- if I go around and shake hands with everybody-- the total number of handshakes will be twice the number of pairs of people that shook hands. No. If we do some set of handshakes-- so that's the graph scenario. So some of us shake hands, and I count how many handshakes I did, and everybody counts how many handshakes they did, we add them all up, it will be exactly twice the number of actual handshakes that took place. That's this statement. That's why it's called the handshaking lemma.

This is good news, because we know this number is 4n minus 6 here. So what could these degrees look like? There's n of them. Number of vertices is n. That's the neat definition of n. So on average what could these degrees be? They have to be less than four.

The average degree has to be at least a little bit less than four, because in summation, you get 4n minus a little bit. So the average degree in the summation must be a little bit less than four. Now, some of them are bigger, some of them are smaller. But if the average is less than four, then in particular, there must be one guy less than four.

Some vertex has degree less than four. So it could either be zero, one, two, or three. Two and three, we're OK. One can't happen, because we are supposed to be minimally generically rigid. If I have a degree1 vertex and some stuff, this guy can spin around. It's not minimally generically rigid. If I have a degree-0 vertex, it can float around. It's not minimally generically rigid, assuming you have more than one vertex.

So I guess this is the situation where you have more than one vertex, otherwise you can't build it from a single edge. Assuming you have-- so I should put n greater than one here. So then you get a degree2 vertex or a degree-3 vertex. You use either property two or property four to make an anti-Type I or anti-Type II operation. You'll

still have a minimally generically rigid graph. You keep going. By the end, you'll have a single edge.

That's the only case when you get stuck, because actually-- see, I lied. Here, I have two degree-1 vertices. That's the only situation where you can have degree-1 vertices. And that's when you get stuck. You have two degree-1 vertices, but then I have an edge. I'm done. Questions? Obviously, I elided a couple of key details here, but other than that, it's pretty straightforward. Yeah.

AUDIENCE: What if you have more edges than you need?

PROFESSOR: If I have more edges than you need, then this does not capture such structures. This is just about minimal generic rigidity. But if I have some graph that is generically rigid, it will be a graph I can build this way, plus more edges. Now, how to identify what edges I should remove so that I could do this Henneberg construction is unclear at this point, and that's the purpose of the next theorem. So good question.

It turns out there's an algorithm, which given a generically rigid graph, will tell you which edges you can throw away, so that you get a minimally generically rigid graph. And that algorithm uses the following theorem. This one is from the '70s by a guy named Laman. It starts the same as Henneberg above. Graph is minimally generically rigid in 2D if and only if it has 2n minus 3 edges. And again, I'm assuming the number of vertices is bigger than one.

So far, this is just our regular degree or freedom analysis. But I said this wasn't enough. You something else. Here's the something else. If I take a subset of vertices-- and let's say there's k of them-- then that induces at most 2k minus 3 edges. So this-- again, minimal generic rigidity. I'm still not directly talking about the bigger case. We'll worry about that later. So there's the fewest possible edges. That means 2n minus 3.

But now, essentially what I want to say is that there aren't too many edges in any one place, and therefore the edges are well distributed. So if I take some subset of

the vertices-- so here's my graph, and I take some crazy subset-- any blob the vertices I want, I look at what are all the edges inside between vertices in that set? I don't want to have too many. I don't want to have more than what I should-- 2k minus 3 if there's k vertices in here.

If I had more-- for example, that's minimally generically rigid. If I did that anywhere in the graph-- this has 2n minus 2 edges. It's too many. Or 2k minus 2 edges. K is four here. If I put that in some bigger graph and the bigger graph has 2n minus 3, and I wasted an edge here, that means somewhere else, it will be flexible. So it's a bit of a weird condition.

You might think this should say greater than or equal to be. It'd be more intuitive. That would be wrong. The theorem isn't true when you put greater than or equal to, because you could choose vertices that have no edges between them. There's going to be a lot of them. But if you make sure there aren't too many anywhere, then it turns out it will be just right everywhere. It's like three little bears or something. So some generalized version of three little bears. K little bears.

[LAUGHTER]

All right. So this is a very cool theorem. It's not obviously algorithmic, because it says, oh, you just check every subset of k vertices for all values of k. There's exponentially many subsets. It's not a good algorithm by itself, but it turns out you can make it into an algorithm. I'll talk about that in a moment.

First, we're going to prove the theorem. OK, I have to speed up a little. All right. Let me sketch this proof. It's really easy, because we already have this great characterization of minimally generically rigid graphs. It's just things you can build by Type I and Type II operations.

So if I want to show the forward direction, that if I have a minimally generic thing-- i.e. It can be built this way-- then it has 2n minus 3 edges. Well, that we already know. And then the other thing is that every subset of k vertices has at most 2k minus 3 edges among them.

So think about it here. So maybe this is my graph. It's produced by a Type I operation at the end. Well, it takes some subset of the vertices. Either the subset contains this vertex, or it doesn't. If it doesn't contain this vertex, then it's a subset just in the old graph. And by induction, my theorem will hold, because that's a smaller graph.

If it's a subset that includes this guy, then it's whatever the subset contains over here, plus 2 edges minus 1 vertex. And that just works out. Over here, there's going to be k minus 1 vertices, so it'll be two times k minus 1 minus 3 at most. We add two edges, we add one vertex. That's exactly what this predicts. You should add two edges for every vertex. So that's fine.

The Type II, same deal. But in general, I'm adding three edges, removing one, and adding one vertex. And that's again true in the subsets, not just overall.

I take a subset containing this guy or not. You have to check all the cases, depending on whether it contains this guy or this guy or this guy, or not. But in all cases, this inequality still holds. How's that? It's a bunch of case work, but all really straightforward, so believe me.

The hard part is the other direction. If I'm told that this is true-- all the subsets don't have too many edges-- then I claim I can actually build it this way, or I claim that somehow it's minimally generically rigid. And we are, in fact, going to build it using Henneberg constructions, using these properties that we proved-- one, two, three, four.

All right. So what do we know? We know that there are 2n minus 3 edges. Hey, I already have a great argument for when the number of edges is 2n minus 3. I do the handshaking lemma, and I know that the sum of degrees is 4n minus 6. Therefore, I know the average degree is less than four. Great. Problem solved. Therefore, I know there's a vertex of degree less than four.

It could be zero, one two, or three. Can it be zero? Hope not. How do I prove it's not zero? I know how to prove it's not one, think. I have to cheat here. Aha. Right, right.

OK, good. Duh. All right. Number of edges is 2n minus 3.

Suppose my graph looked like this, where there's no edges incident to this guy. Well, how many edges does this have? 2n minus 3 still. I didn't remove any. So I've got n minus 1 vertices, yet I have 2n minus 3 edges? That ain't right. So that can't happen.

Similarly here, if I have one edge here, then the number of edges inside this blob-- everything except that one vertex-- would be 2n minus 2, I guess. But it should really have at most 2 times n minus 1 minus 3, which is 2n minus 4.

Sorry, I did this wrong. 2n minus 3. This should be 4. This should be 5. I'll get it right eventually. So if I remove one edge from 2n minus 3, I get 2n minus 4. I had the sign error. But I'm supposed to have 2 times n minus 1-- that's the number vertices over here-- minus 3, which is 2n minus 5. So I have the wrong number of edges. In fact, it should equal. Whatever. At most, at most.

We know in every subset-- here, I'm taking a subset of n minus 1 vertices. I have at most 2k minus 3. Here k is n minus 1. So I can't have degree-1, I can't have degree-0. Therefore, in fact, it's either degree2 or 3. If it's degree-2, I'm done. I'm happy. If it's degree--2, then I do an anti-Type I operation.

So that's this scenario. If I have any degree-2 Vertex, I remove it. And I want to induct on the rest. Now, what do I need to induct? I need to know that this property still holds on my smaller graph.

When I remove this vertex, I want the remaining graph to still have this property. Does it have 2n minus 3 edges, for the new value of n, n being n minus 1 now? Yeah, because I removed two edges. I removed one vertex. So it still has the right number of edges. We've already checked that.

And I need to check that every subset of the vertices over here has at most 2k minus 3 edges. That's still true, because before I had to consider all subsets containing this one or not. Now I'm just looking at subsets that don't contain it. So I still have the Laman conditions holding for the smaller graph in here.

Therefore, I can induct, and we're done. And we're done because then we conclude that the smaller graph is minimally generically rigid. And then property two tells us that this thing's minimally generically rigid, and then I add this vertex, two edges, it'll still be minimally generically rigid. And that's what I wanted to conclude in the left implication direction.

The hard case is degree three. Again, there's more than one Type II operation. I already wave my hands, claiming that there is an operation you can do so that the result is minimally generically rigid. But is that enough? I have to think about it for a second. It's minimally generically rigid.

Ah, it's not enough. It's annoying. That's what I thought. I'm not surprised, but I'm re-surprised. Whatever. For this theory property to hold, I need that the graph I have is minimally generically rigid. That's what I want to prove. I don't know that that's true. So I can't use property four. Sucks to be me.

I'm trying to prove the graph is minimally generically rigid. All I have is degree-3 vertex, and I have Laman's condition. I don't know whether property four applies to my scenario. So we've got to do work. And wow. See what I can do to this proof.

In the degree-3 case, I want to find an anti-Type II operation. Presumably they're out there, but I need a good one. For me, good means it should preserve the Laman condition. So after I do the anti-Type II, after I delete those three edges, delete the vertex, add one edge back, I want this to still hold.

Now, there's essentially two things that could go wrong for this to still hold. One is you go to add an edge, and it's already there. If it's already there, you can't really add it again, and then the number of edges will be wrong. You always want to have 2n minus 3 as you go down. If I try to add an edge and it doesn't work out-- because the whole point is I'm supposed to remove three, add one. If I don't actually succeed in adding one, it'll be the wrong number.

So first thing you need to check is that-- so I take this degree-3 vertex. Here it is. Now it's all about its neighbors, these three guys. If all three of those exist, I would

be in trouble.

Fortunately, that can't happen. This is the picture I drew that cannot happen. This is over-braced. There's too many edges here. This is a subset of four vertices. It has more than 2k minus 3 edges. So by Laman's condition, this can't hold. Some edge must be missing. That's the one we'll target. So at least there's an edge that's missing. That's property one.

There's another thing that could go wrong, though, which is I go to add in this edge, and somehow the other part of the condition-- the subset property-- doesn't hold. Now, what could that possibly look like? It would have to be a subset of vertices.

It should at least contain these two vertices. Actually, it probably shouldn't contain this one. This requires some thinking. I mean, if it contained this one, it's no big deal. I mean, it's got do one more vertex, two more edges. Who cares.

But if you just look at one of these subsets, then in this subset, you're adding an edge. That's in general a bad idea, because if it had exactly 2k minus 3 before and now I add an edge, it'll have too many edges. That would be bad.

So somehow I want to say there's some place I can add an edge where there was slack, where there were less than 2k minus 3 edges. And so when I add this in, I'm OK. It's a little tricky to argue. What we do is suppose-- we're going to argue by contradiction. If this is impossible-- so I'm going to try for these two vertices, then I'll try it for these two vertices, then I'll try for these two-- three choices. If they all fail, I claim there's a contradiction. And I will try to sketch this for you.

So let's say there are Si-- I have some subset of vertices Si-- and i is one, two, or three. And this will turn out to be the bad case. Suppose that it contains-- I didn't give these labels. This is V, V1, V2, V3. So it's going to contain Vi plus 1 and Vi plus 2-- those are the other two vertices from i-- but not V. Not the vertex that I added, because that turns out not to matter.

And suppose that it induces exactly 2k-- what's k here? Size of Si-- minus 3 edges.

It has Si edges. If it has exactly 2Si minus 3 edges, I can't add another one. And if this is true for this pair and for this pair and for this pair, then I'm screwed. So suppose that it's true for all three pairs of these Sis.

Fun fact. Let's look at all of the edges among S1 and S2. Let's start combining these sets in different ways. Speed through this a little. Something called inclusion-exclusion principle, which is I have these two sets S1 and S2. They might overlap, who knows. But if I count all the edges in both of them together, that's kind of like counting all the edges in here-- all of Si-- and counting all the edges in S2-- and then removing the intersection because I double-counted there.

AUDIENCE: It's like probability.

PROFESSOR: It's like probability. This is inclusion-exclusion. It exists all over the place. Usually it's combinatorics. But it's not quite accurate here, and that's why there's a greater than or equal to. Because we're counting edges not vertices, it's a little messier, because there's some edges you miss, like an edge from this vertex over to this vertex. You'll still count twice. But hey, it's close enough. Greater than or equal to. OK?

Now, I claim this thing should only have one vertex in it. How the heck could I claim that? Well, let's suppose that it has at least two vertices. And then I'm going to get a contradiction. So I guess I really mean this-- that Si intersects S2.

I claim these intersections have to be pretty tiny. The reason is, if it weren't, what do I know about the edges induced by S1? Well, it's a subset, so it satisfies the 2K minus 3 property.

So it's going to be greater than or equal to, right? Actually, I'd already assumed the number of edges induced here is exactly twice the Sis minus 3. So this will be exactly equal to 2 times S1 minus 3. It's 2 times S2 minus 3, and then the same thing there. OK?

But this is some subset. As long as it has at least two vertices-- yeah, this should be k greater than or one. Things don't quite work with one vertex, because then it says 2 times 1 minus 3. It should have negative 1 edges. Ain't so. It's got zero edges. So

you need more than one vertex for this to apply.

But now I have some subset. It has at least two vertices, let's say. So it has at most 2k minus 3 edges. So this is going to be at most-- it's confusing because everything's backwards. Well, it's going to be another two times the size of this thing minus 3. So I'm just going to collect them all together. We have S1, we have S2, and the one that we just added was S1 intersect S2 minus 9. It's three 3s.

But this thing-- and I got it wrong. There's a sign error. I got two sign errors. There's a minus here. There's 2k minus 3. And so this is going to be two negative 3s, then there's going to be a negative negative 3. 2 wrongs make a right. It should be 3 in the end. OK.

Now, this thing. What's that?

AUDIENCE: The union.

PROFESSOR: It's the size of the union. Here it's exact, because we're counting vertices instead of edges. It's this plus this minus the intersection from double counting. Huh. So two times that size minus 3. Hmm. So I looked at the edges induced by the union. I got those at least to k minus 3. So what--? Right.

It doesn't sound like a contradiction yet. We want it to be at most 2k minus 3 but I think there's a property I didn't prove yet. Great. All right. So what. Let's look back at this picture.

So if I take S1 union S2, S1 hits these two guys-- 2 and 3-- S2 hits these two guys-- 1 and 3. In their union, they're going to hit all three. So the set is going to look like this. It's going to contain V1, V2, V3, but not V. Interesting.

I know that if I put V in, I still have at most 2k minus 3 edges. If I remove V, I remove one vertex and three edges. Three edges for the price of one. Or for the price of two, I guess. Normally if I remove a vertex, I should remove only two edges. If I cut out V, I remove three edges.

Therefore, this set that includes V1, V2, V3 has to have slack. It can't have 2k minus 3 edges, because-- this should be OK. The bigger set should be at most 2k minus 3. When I remove a vertex and remove three edges, there's going to be too few edges over here. Two few edges is OK. It would be strictly less than 2k minus 3, and here I'm claiming it's at least 2k minus 3. That's a contradiction.

Now, it turns out, I'm not done, because I was in two levels of contradiction. So what this contradicts is this assumption that there are at least two vertices in the intersection. Now, I know there can't be two vertices in the intersection. It's got to be fewer. Could be zero, but it can't really be zero, because S2 and S2 both contain V3. So it's got to be exactly one.

In fact-- it's the crazy proof. When I read this, I was like, really? It works, though. You can do all of them. This is going to equal exactly V1 and S1 intersect S3. You can do this for any pair. So this is going to be V2. So all these intersections have size one, and we know what it is. So what?

Well, now if I take the union of all of them-- there's only this sharing. There's only three shared guys. So I get twice the size of all them, which is S1 plus S2 plus S3 minus 3. Not twice. Sorry, just that. Who cares. Take all the edges induced by those guys. At the very least, that's going to be-- good. Now I see. I think that'd actually be equal.

But the point is they don't share any edges. Because they only share a single vertex, there's no room to share any edges. So these guys should be probably equal to each other. Edges induced by all of them versus edges induced by each of them individually. Here the whole is equal to the sum of the parts.

And this is going to be 2 times number of S1s plus number of S2s plus number of S3s minus 9. This should be equal. We assumed that they were exactly the right number of edges.

So this is the usual problem. This can't happen by the same argument we did here. If we include V1, V2, V3 and we don't include V, we've got to have too few edges.

And here we're saying we have exactly the right number. Can't happen.

AUDIENCE: [INAUDIBLE]

PROFESSOR: It's a slight discrepancy between the 9 and the 3. Sorry, question?

AUDIENCE: [INAUDIBLE].

PROFESSOR: There's no 2 here. Here I'm counting vertices, here I'm counting edges. So I get a 2 here. This thing is equal to this thing plus 3. Ah, algebra. I multiply the 2 by the 3. I get 6. It cancels with the 9. I get minus 3. So this says I should have exactly the right number-- twice the size of the set minus 3.

That can't happen. I have to have at least one fewer. That's what we argued. It's messy. You should read the notes. But that proves Laman's theorem.

Now, in the remaining four minutes, I want to tell you two more cool things. Three more cool things. Great. All right.

First thing is you can turn this into an algorithm. How do you turn it into an algorithm? It's a crazy idea. Roughly speaking, in every vertex, you imagine there's two little pebbles sitting on the vertex. And then there's a bunch of edges incident to that vertex. You can assign each of these pebbles to one of those edges. Something like that. You get to choose each, but I only have two of them. I do that for every vertex. So something like that.

If I can cover all the edges-- there's a slight extra condition-- but if I can cover all the edge in this way, I claim I'm rigid. Well, not cover all the edges. If I can get all the pebbles off the vertices without them hitting each other, then I'm rigid. I might have more extra edges that aren't covered. Those are the extra edges. You should throw them away. That's over-bracing. It's too much to be rigid.

But if I can get all these pebbles off somehow, then I have roughly the right number of edges for vertices. And while it's not obvious, it turns out you will satisfy Laman's condition in that situation. And so it turns out to just be a graph searching problem.

You have to push the pebbles around in order to get them off the vertices. You don't have a lot of choices where the pebbles can go. And it turns out to be basically m calls to path connectivity operations.

So I've just given two graphs. So I'm given a connected graph, which is the ways that I can move the pebbles around. I just want to know, can I do a chain reaction of pebble twitches till I get this one moved away? So it's just a depth first search. Whatever.

So this cost linear time. I think we could probably get away with n operations each linear time in number of edges. So the total amount of time is number of vertices times number of edges. M is the number of edges. Polynomial time. Life is good. Open problem. Can you do better? But that's been around for a long time. We've tried it before. Seems quite challenging.

Bigger open problem. What about 3D? In 3D, there these annoying situations, like this one. Two tetrahedra glued along a face. Take two-- we call this a banana-- take two bananas. Joined them at vertices, the opposing vertices.

This thing is flexible, right? You just rotate through here. It's generically flexible. It doesn't matter how I draw it. You can rotate through that axis. But it has 2n minus 6 edges, and every subset has at most 3n minus 6 edges. So Laman doesn't hold.

Does the generalized version of Henneberg's theorem hold? We don't know. I think maybe yes. If you want to work on 3D rigidity-- which is a little scary, because a lot of people worked on it-- I think Henneberg might be a good way to go. It might not lead to an algorithm, but at least you might get a nice characterization. It'd be fun. I think Henneberg would make a great puzzle. By working forward not backward, it's quite challenging.

Other fun things. Let me tell you briefly about polyhedra. Polyhedra are fun, and they exist in three dimensions. Here's a polyhedron. It's a convex polygon-- it's Buckminster Fuller Geodesic Dome in Montreal-- and it's convex. It's made of triangles.

Is that rigid? It looks like it. Hasn't fallen down yet-- it's been there for 40 years now. Indeed, you can prove that thing is rigid, and this is the basis for geodesic domes. Turns out, if you have a convex polyhedron-- so 3D I said is really hard. But this does not look like a convex polyhedron.

So what if you had a 3D linkage that is the surface of a convex polyhedron and every face is a triangle? I mean, ignoring the floor for a little bit, but you can deal with that too. Every face here is a triangle. It may be easier to see in this diagram.

So it's a linkage. It's a bunch of triangles. That thing willl be generically rigid. And even rigid-- never mind generically. You can prove this thing is rigid, and it was proved, roughly speaking, by Alexandrov, let's say, or Den-- the early to mid 1900s. That's cool.

And that doesn't mean that there aren't other configurations. So here's a convex polyhedron, and you could triangulate the sides also. And there's a non-convex realization of that. But as long as you stay convex, and if you move-- this requires a big jump. And you can show that if you have a convex polyhedron, in order for it to flex continuously, well, it can't. In order to flex at all, it has to jump into a non-convex state.

But there's a fun thing that happens when you look at non-convex polyhedra. And it's really hard to see this polyhedron, this dent on the backside, but you can cut this out and make one. And it's a non-convex polyhedron, and it does have a continuous flex. It's a flexible linkage. It's triangulated, but it's not convex, and so Alexandrov's theorem doesn't apply. And you can move it continuously. It's a flexible linkage.

Fun fact-- when it moves, the volume stays constant. It's called the bellow's theorem-- it was open for many years-- by Connelly and others. And so in fact, you take any-- not just this polyhedron-- you take any polyhedron that can flex-- it has to be non-convex-- its volume will stay constant throughout the whole motion. So if you've ever played with the bellows, you may think this is not true, because bellows pump air in and out somehow.

And the theorem is well, you can't build a bellows out of linkages. You could maybe build bellows out a paper, or a flexible material, but if you cannot triangulate that surface, you cannot make it a metal bellows. Never mind if they exist in reality. They are theoretically impossible to build. I don't know-- are there metal bellows?

That's it. And next time, we will talk about more rigidity and things called tensegrities, which even more exciting. And we'll see how this ties into actually getting things to fold from point A to point B, not just do they fold at all. It's all related.