PROFESSOR: Welcome back to 6.890. Today, we will do many more hardness proofs, mostly of games and puzzles and some graph theoretic problems, all reducing from 3SAT. We're going to continue where we left off last time, which was with this picture, which also comes from lecture 1, which is how to prove *Super Mario Brothers* is NP-hard. We had two main gadgets, the variable gadget, which forces your player to choose going left or right, and the clause gadget, which could be visited in three different ways in order to enable it. These each contained an invincibility star, so that if later, you come through to traverse the gadget, you can get through all these fire bars if and only if there was at least one star there. So that was a regular 3SAT clause.

And then, there's the connectivity in between. And so in particular, you need a crossover gadget with this set-up. But the general idea is you visit variable, visit all the clauses that are in it that include that literal-- you get to choose only one-- then you go to the next variable, next variable. And then, you traverse all the clauses and you can get to finish if and only if the 3SAT formula was satisfiable. This is a general construction or approach for platform video games. That's what we were presenting it as in a recent paper.

I thought I'd show you a few other Nintendo games you can prove hard with this technique. This is the clause gadget for *Super Mario World*. So in *Super Mario World,* there are these great question-mark blocks and in particular, POW blocks, which go back to the original *Mario Brothers* without the "Super." How many people played *Mario Brothers?* Really old-school, pre-*Super Mario*. And anyway, it doesn't matter exactly where it comes from. It does something completely different in *Super Mario World*, which is turns all coins into things you can walk on for 10 seconds.

But you can't normally hit a question-mark block except from the bottom. There's no way to do that here. But if you can visit one of these three places, you can send one of these turtle shells flying and it will hit this and bring the POW block off and then disappear, so that later when you try to traverse this clause gadget, if there's a POW block, you can turn these into solid things for long enough to get across but no longer than that. So in the next gadget, you need another POW block, and so on. So that's clause gadget for *Super Mario World.* The other gadgets are pretty similar, so I'll just show you that one.

AUDIENCE: Does the star trick not work?

PROFESSOR: I guess you could also do the star trick, but I think we drew this figure before we did the star trick. So we thought, oh, this is cooler.

Now, one thing I thought I'd mention briefly is some fun glitches you can do in the real *Super Mario Brothers,* as implemented on the ROMs. You can do these crazy things like jump off of walls. You're not supposed to be able to do this and this level would be unsolvable in the way you would normally think about *Super Mario Brothers,* but there happens to be this fun subpixel resolution. I think this is usually very hard to do with a real controller, but much easier to do when you can slow things down 100 times and every frame, you have a choice of whether to push a particular button. That's called tool-assisted speed runs, usually.

I don't know if you consider this real *Super Mario Brothers* or not. It depends on your notion of reality, whether you're more of a mathematician or a gamer, I guess. But you can try to modify the gadget so that wall jumping doesn't help you. So you could imagine trying to jump up these walls. But if there's things in the way, it prevents it. So if you just add that to every vertical tunnel, that will prevent going back up the tunnels, which is important for the variable gadgets. We don't want to be able to reverse our decision and later choose the opposite variable setting.

Here's another fun set of cheats where you jump through walls. This one is apparently somewhat doable on a real controller, but still quite challenging. I think that one of the general ideas is the collision detection is not super good in the game

as implemented. So if you get just the right geometry, you can walk through the walls. There's instructions for you.

So this is obviously problematic because all of our gadgets involve walls. So if you can walk through them, not very interesting. So a general approach is to just add lots of monsters into the walls. So even if you could walk through, you become small Mario and then that's not an option. I should say these have not been thoroughly tested, so it's plausible you can walk through the wall in such a way that maybe you kill the monster or there may be other glitches that we don't know about. And so you might consider this a work in progress.

But generally, I would believe you can modify the proof to work on real *Super Mario Brothers* as implemented in real physics, of course, generalized to an n by n screen and so on. What exactly you call "real" is challenging there. But that's *Super Mario Brothers*-- just some more fun stuff in the paper.

Let me talk a little bit about *Legend of Zelda.* This is what it looks like, in case you've never seen it. And this is the original *Legend of Zelda.* You're a little character. You can walk around. And this is what I would say is a typical block-pushing puzzle in *Legend of Zelda,* which is try all the blocks and hope that one of then pushes. And I have to say, I watched an entire-- I don't know how long it was-- five hours of walkthrough of *Legend of Zelda* wanting to find whether there are any more complicated block-pushing puzzles. I swear I remember a slightly more complicated one, but I could not find it. It's possible it skipped in this particular walkthrough.

Well, it just takes a long time. But you get it. All right, we're almost done. You get an idea of what the game is like. You've got a sword, whatever. That won't be particularly relevant to our reductions. There will be no monsters in our reductions. In this case, I think you just have to wait a really long time or possibly kill the hand that comes out. Then, you can push that block. He actually tried to push it in the beginning, but it wouldn't move. Anyway, that's *Legend of Zelda.*

Here's a slightly more interesting one. This is what I remember. Normally, you push one, two, three, and then you push this to the side. In this particular case, blocks

are not pushable sideways, which is really annoying. So you actually have to solve it a different way. If you watch this video, at this point, he's like, oh, my god. This was the simplest block-pushing puzzle ever and I solved it. Yea! It's after half an hour of trying other things. That's the joy of *Legend of Zelda,* very fun puzzles, so to speak.

So the point is, you have block-pushing puzzles. We already know-- oh, sorry. Here's one more, I guess. These are in *Legend of Zelda-- A Link to the Past.* So it has a slightly more com-- there's an actual slightly nontrivial block-pushing puzzle. But in general, we know push-one is hard. Most versions of *Legend of Zelda* are slightly more restrictive than push-one. I would call them "push-once" blocks, where you can only push one at a time. That's your strength. But also, you can only push a block once. You can never push it a second time or push it back or do anything like that.

And with some slight-- this is the gadget we saw before. With some slight tweaks, you can make the locks still work when you're only allowed to push each block once. So that proves that most versions of *Legend of Zelda* are at least NP-hard, because they include block pushing. But it would be nice to understand other senses in which these kinds of games are hard without using block pushing.

So here's another feature called the hookshot. So you select it and then if there's an object on the other side, you can hookshot to that side. So you can't walk over the cavern, but you can hookshot across. Here it is again. As long as there's an object over there-- you can also pick up objects and destroy them. So does that make it hard with no pushing blocks? Well, a little bit of pushing blocks.

Here's a simple proof using that structure. So here's a variable. I go from top to bottom. You see over here a treasure chest or something that you can hookshot onto to go over the cavern. And here, you have a choice. You can go left or you can go right, but then you can't back up, because there's no object here to hookshot onto. You can only do it orthogonally. Here's a clause. If you come down any one of these paths, then you can push this block one step. And then, you'll be able to hookshot from the right to get onto this platform and then continue to the left. And a

crossover is really easy in this world.

So you could come up with lots of different NP-hardness proofs. So this is a fun one for *A Link to the Past,* which is the first game that has that hookshot. This is *Metroid.* If you've never played, most of the time you spend going up and down elevator shafts. And there are these monsters which you can shoot. And then, there's also this ability to become a one-by-one block instead of a two-by-one block and roll through small things. When you're in that position, in a certain part of the game, you are not allowed to shoot. So that's the situation we're going to imagine.

So here is a clause in a crossover. The variable gadget is the same as *Mario.* You fall left or fall right. You can't go back up, because you have a limited jump height. So let's do the clause first. There are three different ways to visit it. When you're setting your variables, if you can get here, you can aim upwards and just shoot all of these things. It would take a few minutes, but you'd get rid of all these guys. These guys just walk around constantly.

And then, to traverse, you're going to be coming through here in ball mode. There's no way to-- I guess, here, you could briefly stand up, but you can't shoot down in original *Metroid.* So you want to roll over here, but you would take too much damage. You maybe only have one hit point left, so if there's any of these guys left, you have to visit at least one of these to kill all the monsters and then get through. That's clause.

Crossover is a little more challenging. In this case, we set things up-- the idea is you come in here and you want to leave out there. So it's uni-directional. You can't go up in this gadget. And the idea is there's a little gap and it's moving this way as the monsters move around. So you wait for the gap to come here. Then, you go down and then, you're basically forced in timing when you fall down this cavern.

And it's set up so that by the time this thing gets to here, these guys will have come to the other side. Then, you have to roll to the right in order to avoid taking damage from these guys who are going around the other way and similarly in the other direction. So you have to get the timing right and we haven't exactly tested this yet,

but it should be approximately correct. And that will force a crossover. Should be a simpler way, but that's the best crossover we have currently. So *Metroid* is NP.

Here, we actually claim NP-completeness, because there isn't too much state to the game. So the issue here is the length of a solution polynomial length. If it is, then it's in NP and NP-complete. And here, you can kill each monster only once. So that's the idea for-- and the amount of state you have is just your position and what items you currently have. So you should be able to solve this in NP and that sort of settles the complexity. Most of these other problems could potentially be harder than NP and we have some other results like that, which we may get to when we're talking about PSPACE. But today is about NP.

Here's *Donkey Kong Country.* If you haven't played, this is a speed run and you can kill monsters by rolling. Most of the time, you're jumping and collecting bananas. There are these bees that if you touch them, you die or you sort of have two hit points. And then, there are these barrels you can pick up-- that's a giant bee-- you can pick up certain barrels and then throw them at a monster and, in this case, damage, usually kill them. Then, there are these other types of barrels that when you touch them, they shoot you in a predefined direction and let you collect lots of bananas. So those are the rules of *Donkey Kong* in a nutshell, in a banana shell.

AUDIENCE: In a banana peel?

PROFESSOR: Sorry, in a banana peel, thank you. So here's a simple crossover with that setup. You just have these barrels that shoot you in a predefined direction and force you to go through this intersection without any choice. When you're being shot, you cannot move. So that's easy. Again, variables the same as *Mario,* you fall left or fall right. And a clause-- it's a little hard to see with the background here, but there's a little ledge here. And you can come down this way, pick up this barrel, and throw it over here. And it will eventually hit this bee, so that later, you can traverse this gadget without dying from the bee.

So not drawn is a platform that you need to-- or maybe you're falling all the way through these things if you set up all the clauses vertically. And if there's any bee

left, you will die, otherwise not, and so you have to visit at least one of these three barrels to kill the bee. And I think this is a mechanism to prevent you from carrying this barrel anywhere else, like climbing up or something. Anyway, that's *Donkey Kong Country.*

I think there's one more, *Pokemon*. I'm kind of embarrassed. I've never actually played a *Pokemon* game. I don't know if I should admit this, but I need to sit down and play them all some weekend. I did watch Twitch play it for a while.

AUDIENCE: Going back to the previous slide, in some of the other gadgets, if we satisfied a clause, there was no way to-- there was a barrier between setting the clause and going through the rest of that thing. So potentially, you could get to the last clause here, kill the bee, and just walk forward without-- can we with the rest of the stuff?

PROFESSOR: Good question. Yeah, I think probably the right way to set this up-- I'm not quite sure how to do it directly from the drawing, but an easy way to do it would be there's no floor below here. And at the end, when you solved all the things, there is one of these barrels that shoots you straight to the left until you get to the finish.

AUDIENCE: Ah.

PROFESSOR: So if you tried to exit the clause early, you would just fall to your death, because there's only one barrel shooting way over on the right. That would be one solution. I forget exactly what's in the paper, but it wouldn't surprise me if it's that. Yeah, good question. You definitely don't want to be able to just visit the last clause, kill the guy, and then exit. That would be cheating. But in these proofs, that's always what you have to worry about is this type of cheating. Other questions? Cool.

Pokemon time-- so *Pokemon* is a somewhat complicated game. It's a lot of different abilities and so on. So we had to abstract things away and in these pictures, there are two types of-- I haven't defined any of the rules yet. You are a little person walking around here. You're carrying Pokemon, whatever. And there are these characters called trainers and the red trainers are called "weak" trainers. These are people that when you visit, they just sort of deactivate. You beat them. And then,

there's blue trainers. We'll call them "strong" trainers. Those are ones that no matter when or how you try to play them, you always die. And so you fail, so you never want to visit a blue trainer.

Now, trainers have two kinds of-- let's talk about variables maybe. This is an easy situation. So a variable, you want to come in the "in" and either leave in the "false" or leave in the "true." And there are two ways to activate a trainer. One is to walk into their field of view, which are drawn with these rectangles. If you walk to this position, the trainer will walk to you and then you fight the trainer. And because it's red, you'll win. And so in that case, the trainers moved here. Trainers are obstacles, so you can't get to the true anymore. You have to go around to false.

The other way you can visit a trainer is to talk to them from the side or from behind. So you can walk here, talk to the trainer, beat them, then they deactivate. Then, you can walk this way. And you can only fight them once. Then, it's over. They won't move anymore. So that forces you to do true or false and then commit to that decision. And these are obviously obstacles that are immovable.

All right. So the idea with the clause is the following. When you're traversing the clause at the end, you want to walk through here and leave. And the worry is this position. If this trainer can come to you, you will die. So that will be possible exactly when all of these guys are out of the way. When will they come out of the way? When you walk through them-- and none of them have been activated yet. So then, they will walk one space to you. You'll win, but it's temporary. You win the battle, lose the war, because when you get here, then this guy can walk through.

But these serve as obstacles otherwise. So if you can come down one of these paths and visit the trainer early, then later when you come through, they will not move toward you. They will stay an obstacle for this guy in, so this guy cannot walk to you. This is just to make a barrier. And then, you can get through. Crossover, you get the idea, right? We could try to walk through it, but it's definitely complicated. In general, we want to go from x to x prime or y to y prime. Again, it's directional.

And now that I know the right color assignment-- red is weak-- I would guess this

guy is an obstacle. But as soon as you go this way, this guy moves out of the way. So it kind of closes off trying to return this way. So once you come through here, this would be a problem to come back. So that kind of cuts you off from going back and it's kind of like the crossover we saw with push-one 2D, I believe. Then, you probably go here. A similar thing happens. This guy will approach, come to you, and then you can escape. When you get here, this guy will come all the way down. Oh, that looks bad.

AUDIENCE: Can you [INAUDIBLE]?

PROFESSOR: Yeah, so probably we should have talked to him early. So then, we can come through here without any trouble, succeed in getting down to y prime. You have to check that you can't get to x prime. This looks like an impossible nexus to get through, but these guys can walk through. That's the reason they're there instead of just being obstacles. Left to right should be-- sorry?

AUDIENCE: They can only see you five or six tiles away.

PROFESSOR: This is a generalized *Pokemon.* Yeah, it would be nice. Obviously, this is a bounded visibility. It would be nice to get that down smaller, but at least this shows constant visibility is enough, because these gadgets are all local. It would maybe also be nice. Are they all the same visibility?

AUDIENCE: Yeah, it's just some fixed amount of tiles.

PROFESSOR: So it would be nice if they were also all the same. That does not currently hold, but it wouldn't surprise me if that-- well, yeah, tricky. All right. I'll just mention for a little bit of *Pokemon* cred, this is one paragraph of our paper where we construct the weak and the strong trainers using exactly the setup that's available in various *Pokemon* games.

I believe in every single game there are enough abilities that you can set up so that you either force the bad guy to all he can do is self-destruct or all he can do is kill you and you lose, that sort of thing. But it requires some *Pokemon* expertise which Alan Guo had in this case. He's a PhD student in C-Cell here. Any other questions

about Nintendo games before we move on?

AUDIENCE: This is a common sort of *Pokemon* specific. In Generation One, moves can miss with a one and two [INAUDIBLE] even if their accuracy is 100%. Are you good with this?

AUDIENCE: It's self-destructing.

AUDIENCE: [INAUDIBLE].

AUDIENCE: The self-destruct means he doesn't die.

AUDIENCE: No, the self-destruct doesn't matter whether he does damage, right? It's just self-destruct, you die.

AUDIENCE: I think self-destruct, the death doesn't actually happen if the attack misses. Can you?

PROFESSOR: No, it does. It does.

AUDIENCE: Really? OK.

PROFESSOR: Yeah, it does. We'll have to go play.

All right. The next game I'm going to talk about is a sort of physical game that you play on usually a go board with go stones. So they're white stones and black stones. It was invented by this guy, John Conway. And so if you've ever played Conway's Game of Life, that's a zero-player game. This is an actual two-player game and quite challenging.

Here's the general idea. There's a right player trying to get the white ball-- that's the "football." This is philosophers' football, if you will. You're trying to get this ball over to that goal and the left player's trying to get the ball to the right goal. Then, there are these black pieces, which are called "men." Those are the players of the game, so to speak. And there are two types of moves in this game. One move you can do is just place a black stone anywhere on the board that's currently empty.

And the other type of move is the more complicated one. This is called "kicking the ball." You can take the white piece and you can jump it over a string of consecutive black pieces. Those pieces are immediately removed, so this gray shading means they've been removed from the board. But in the same move, you can do several such jumps. So I can jump here. Then, I can jump here. Then, I can jump here. Then, I can jump here. And everything I jump over-- in this case, I won. If I'm the right player, I got to that position.

So this is why the right player-- from the original position, maybe they wanted to place this stone because that would enable in their next move-- you can't place a black stone and then jump. You can do one or the other. So you could try to place this position, place this stone hoping that in your next turn, you'll be able to do this jump and win. But instead from this position, the left player could say, oh, well, I'll just add this stone and then in my next move, I'll be able to jump to the right side. And that's, I think, unblockable. So left wins. Yea!

So that's the setup. This is a fairly complicated game. You can get to the same position multiple times and so it's actually open how hard this game is. I think there's an upper bound of it being solvable in exponential time, so it's in EXP.

AUDIENCE: What do you mean by "solvable?"

PROFESSOR: So given a position, you want to know whether left or right will win from this position.

AUDIENCE: Is it possible that it doesn't terminate?

PROFESSOR: I could be a tie is the right answer. So that's tricky. There is a piece-base hardness result relatively recent, but an old result of ours with David Eppstein is that just determining whether you can win in one move is NP-complete. So that's a relatively clean thing. You don't have to worry about non-termination or that sort of thing. It's just one player's move. Here, everything is reversed. The ball is black in this case. I'm not sure if there's a consistent notion, anyway.

And this is actually a reduction all on one slide. It's pretty simple. So this is what we call the mate in one problem. And you can take lots of different games and make

them really clean, well-defined puzzles if you just ask, can I win in one move? Even games that involve cards and randomness and weird stuff. And one movie usually doesn't involve those at all. Anyway, here, we're getting rid of the loopiness of the game and not having to worry about that.

So there's sort of two parts to this proof, the variable traversal and the clause traversal. But here, we have a kind of very different way of connecting things together. We have a giant matrix, if you will. We're starting up here and we have a choice whether-- if we want to set x1 to true, we're just going to go to the right by a sequence of jumps. We jump here, then here, then here. Every time we stop, we have no choice. If we want to keep going, our goal is to get to somewhere, one of the edges of the board, I think the bottom edge. Bottom edge is the one that's hard to get to.

So every time we jump to the right here, we don't have a choice. We have to keep jumping to the right. But over here, we have a choice. We could go down and then jump to the right and get over here and that will do different things in the middle. In general, that is our setting of x1. Then, we're going to do the same thing coming back for x2 of two choices. Then, the same thing going back for x3 and then we get to this corner. So that's forced other than these two choices, binary choices.

Now, we have our first clause. Let's say x1 negated or x2 negated or x3 negated. We want one of those things to be true. And so the idea is we're going to choose whichever one has been set correctly and then we will go up that channel and get to the top. Again, it's pretty much symmetric. So every time we jump over a bunch of pieces, whenever we stop, we can't go left or right. We have to keep going up. So we have to make it all the way.

In fact, each of these columns will be completely filled except for one intersection, which is the thing that makes it false. So here, we want x2 to be false. So if we happen to follow the true path where x2 is true, we won't make it because this piece will already have been removed when we came through here. And so if we try to take this path, we'll get stuck here. There won't be any piece above us. As long as

that didn't happen, as long as this row was not chosen, then we can take this column and just make it all the way up to the top.

So we're going to be able to take one of these three paths if and only if at least one of these things is true. That's the first clause. Then, we do the same thing for the second clause. And if all of the clauses are true, we will be able to get here and then we can jump down and get to the finish line and win in one move, one move. So mate in one is NP-complete.

I have one other example of a mate in one result. It's actually not a hardness result, but it's kind of fun, so I thought I would talk about a non-hardness result for once, which is checkers. Checkers is another game where in one move, you can do a lot of jumps. Especially if you have a king, then you can potentially win by killing a lot of pieces all at once. And this is considered back in 1978. This is the same paper that proves checkers is EXPTIME-complete, which we will get to at some point. But for now, let's think about the mate in one problem. This turns out to be easy, unlike phutball.

So checkers-- I assume you've all played checkers. You can only move diagonally on the black squares. So you can recast that into an orthogonal problem where the pieces can only move orthogonally up or right or vice versa, except king pieces. They can just move orthogonally. And the interesting part is when you can jump over a piece of another color. So if there was a black piece here, this one can jump over. And in general, if you have a sequence of jumps you can make, then you can do all of them at once.

So the problem looks something like this. If you have a black piece and let's say a black to move, if there are a bunch of white pieces, you can jump over them. And the key thing to notice-- again, as soon as you jump over a piece, it disappears. So you can reuse it in the same move or I guess you can't reuse it in general. The key thing is that the jumps preserve the parity, the even or oddness of both your x-coordinate and your y-coordinate. So you're moving around on this reduced grid.

And so you can take a picture like this and turn it into a graph where the potential

positions for you, which are drawn with this kind of pattern, same parity of rows and columns. You could make those the vertices of your graph. And then, there's an edge between them if there's a white stone in between that you could jump over. So then, given this graph, the question is, can you visit all of the edges of the graph with a single path?

And that is the Euler tour problem or the Euler path problem. And it's easy to solve that in polynomial time. You look for odd-degree vertices. There should be at most two of them and you better be at one of them. Or there's zero of them. Then, you're happy. So that's how to solve mate in one in checkers, in case you ever get to those difficult checkers endgames. And now, you can decide how to win or not in one move. Two moves is left to you.

All right. So next topic is called cryptarithms. This comes from the recreational math world and they're kind of fun. You've probably seen this or you may have seen this one if you've seen any cryptarithm. It's a classic. It goes back to 1979 from this book by Madachy.

And the idea is this is an arithmetic formula, SEND plus MORE equals MONEY. And each of these letters represents a digit zero through nine. And so these two M's represent the same digit and so on. Two n's represent the same digits. Those two O's represent the same digit. And furthermore, M is different from O is differ from R. So it's a bijection between the letters that appear in this puzzle and some subset of zero through nine.

And it actually has a unique solution. This is it. You could verify it, but I'll leave it as a puzzle so don't look at that too long, in case you want to solve it. It is doable, but the tricky part are things like the carries in the addition, which make it not just a simple linear system to solve. And in fact, these puzzles, generalized to an arbitrary base, are NP-complete. If you do it in base 10, then there's only at most 10 factorial possible solutions. So that's constant and that's considered fast. And indeed, in real computers, you can solve 10 factorial reasonably.

So for generalized base, here is a proof. This is actually David Eppstein, who was a

co-author in the previous result. This is his first theory paper, "SIGACT News." And it proved that it's strongly NP-complete, meaning even if the base is polynomial, the problem is NP-hard. And it looks really messy, but actually, it's fairly simple.

So let me start with this gadget. This is an easy one. So there's no notion of zero or one in this gadget, but zero is just a letter and one is just a letter. But with this gadget in place, 0 is 0 and 1 is 1, because 0's the only possible digit when you add it with yourself, you get it again. It is the identity in additive ring. So that forces 0 to be 0. And then, here, we're adding p with p. P are just arbitrary things. We sort of don't care what they are. We get something q.

And over here, we add 0 to 0 and we get 1. That's only going to be possible if there was a carry in this column. And carries can be only 1. They can be 0 or 1. So that forces-- we know it's not 0 because 1 is different from 0. Therefore, 1 is 1. The symbol 1 is the value one, the number one.

All right. So that gives us some infrastructure. And the zeroes are really helpful, because it lets us space out gadgets and guarantee that there's no carry between them. This is supposed to be a variable gadget. There's vi and there's vi bar. So let's see what happens. Let's do the vi part first. So this formula says that bi equals twice ai. And there's guaranteed to be no carry because of this.

Now, this is a little tricky. We add yi to yi until we get zi. So there may be a carry or not here. Then, we add bi to bi and we get vi. So vi is going to be 2 times bi plus either 0 or 1. That's going to be our binary choice. We can set up the yi's to either carry or not. We'll call that C. In this formulation, C is the carry in that operation. And bi is 2 times ai. So this is 4 times ai plus C, which is 0 or 1.

And what we're going to think about is modulo four. This is congruent to C. So we're going to treat all of our variable assignments modulo four and they're always going to be 0 or 1. vi became this and the claim is that this part makes vi bar the opposite. And you can work through the arithmetic as I did in the notes. You end up vi bar has to be 4 times something, which is Ci.

So we don't care what that is. We'll set it at the very end, plus 3C plus 1. C is 1, this is 4, and this becomes 0. If C is 0, this becomes 1 mod 4. So in general, it's 1 minus C. It's the opposite choice from what vi was that becomes vi bar. So that's cool. We've got vi. We've got vi bar. We can duplicate our variables for free, because we can just use that same letter vi in many different places and then we just need a clause gadget.

And so the work is similar kind of tricks. We add fi to itself, we get gi. We add gi to itself, we get hi. So at this point, we have 4 times fi plus possibly one carry. This wi might carry or not. So we get hi is 4 times fi plus 0 or 1. Then, we add hi to 1 and we get ti, plus there might be a carry from this column. So that's ti is going to be hi plus 1 plus a potential carry. hi is this thing, so we end up with 4 times fi plus 1 plus 0 or 1 or 2, depending on how many carries we've had total, which is 4 times fi plus 1 or 2 or 3. And modulo four, that is one or two or three.

This is going to be the number of true variables in our clause, because in the end, we just add vi plus vb. We get this thing. We add vc for our three variables, a, b, and c. We get ti. And ti was that thing. So if we just think about things modulo four, that means va plus vb plus vc should be 1, 2, or 3 mod four.

Now, the reason we have this junk-- and the junk's-- well, OK. There's the not modulo four part, the part when you divide by 4. That's sort of tricky. But at this point, at least, you should be convinced that if there's a solution to this problem, then there must be a solution to the original 3SAT instance. The reverse is less clear. If you have a solution to the 3SAT instance, you have to be able to set all these variables to make everything work out. That's a little more tedious.

For fun, and this is an old proof 1987 and I was just looking at it again today, I think, or I think it was yesterday. I think I can simplify it a little bit. Instead of using regular 3SAT where you have all these choices, one, two, or three things are true, if you use exactly one 3SAT, one in 3SAT, it should be a lot easier. So gadgets are smaller. With one in 3SAT, we don't need negation. So I just need the vi part, which we did explicitly over here. So that's kind of nifty.

And for exactly one 3SAT, we don't need to have all those carries. We just need to build four times something-- sorry, that should be fi here. We add fi to itself. We add gi to itself. So here, we have 4 times fi. And then, we add 1. So this ti will be one mod four. And then, we add the variables up and we should get exactly one mod four. Now, admittedly, I haven't checked the second half of the proof with this construction, but it seems plausible this would be a somewhat simpler proof. So it shows you the power of all the different versions of 3SAT we saw last class.

Now, I can tell you a little bit about the other direction. I have some of the details written down here. So the other direction is we need to check if there's a satisfying assignment to 3SAT or one in 3SAT, that we can actually construct these numbers that the base is only polynomial. The trickiest part is to get all of these numbers to be distinct from each other.

AUDIENCE: What base do you do?

PROFESSOR: The base will be something like n cubed, it turns out. And I'll tell you, one trick is we have all these ai's, bi's, and ci's. We can guarantee they are all different from each other, that all the a's are different from all the b's are different from all the c's by saying all a's-- let me get this right-- all a's will be 2, 34, 66, or 98 mod 128. And in general, those are particular numbers set up to make this construction work out. I didn't check them all but I believe them.

In general, we're going to distinguish all the letters by what they are mod 128. So given a number between zero and 127, there's a unique letter of the regular English letter alphabet, the a's, the b's, the y's, the z's, and so on that it is assigned to. And I'll tell you in particular, the vi's and the vi bars are going to be, I think, 8 or 9 mod 128. And the 8 is if it's false and the 9 is if it's true, because remember, modulo 4, we want this to be zero and we want this to be one.

So there's a bunch of choices like that. Some of them are unique. We can always guarantee-- I forget which ones-- the ci's are all something mod 128. So the mod 128 will tell you sort of what letter of the English alphabet it is and then it's a matter of deciding the high-order bits, your number divided by 128 and take the floor. What

that is we set to make everybody distinct. And the heart of the proof, I will say, is to make sure that the vi's are all different. Or let's say that the ti's are different.

So we have the variables here. We're adding them up and we get ti and we need all of the ti's-- there's one ti per clause. We need them all to be different, which means no matter which triple of vi's we get, we should get a different sum of the triples. So this is a problem of set vi and vi prime divided by 128 so that vi plus vj plus vk are all distinct.

And at this point, Eppstein says, oh, there's a result by Bose and Chowla from 1959 the says you can always choose these guys-- if there's n of them that you need to choose, then n-cubed different integers suffice to make this true. That's if you know the literature super well, but there's an easy proof that polynomials enough here. And those of you who are in 6.851 should be able to construct one, because this is essentially fusion trees.

But in general, imagine that you've chosen by induction I'll call them v less than i. And let's ignore the primes. Let's just say we've constructed all the things up to the vi minus 1. And now, we want to choose vi and it needs to avoid certain values. I think if it avoids vj plus vk minus vl minus vm minus vp for all j, k, l, m, p less than i, then no triple should sum to any other triple, because if we add-- I think maybe I have the signs slightly off. So I wanted to do this. Yeah, so this is one more plus.

So suppose I had an equation like this. I claim that's bad because if I move these guys over to the left-hand side, that's a triple of v's that sums to another triple of v's. So as long as I choose my new value v-- sorry, this should vi. That's the one I'm choosing. As long as I choose vi to be different from all such sums and they're only less than n to the fifth such sums, because there are only n of these different things, then I'm OK.

And so by the pigeonhole principle, as long as I have a range that is at least n to the fifth in size, then I can always choose vi to avoid all those conflicts. So that's a really easy proof that n to the fifth is enough. In fact, n cubed is enough, but this is a hardness proof. All we need is strong hardness. We just need n cubed. That's just

fine or n to the fifth. Any polynomial would be OK. So that's a sketch of the reverse direction for this proof. There are some more details to make sure everything adds up right mod 128 and that you can make all the ai's different and so on, but the hard part is the vi's, which seems believable. Questions about cryptarithms?

Cool. Next is an origami proof. So here's the motivating problem. I give you some crease pattern, which is just a graph drawn in a plane, no crossings, and all the edges are straight segments. And you want to fold it into a flat origami or a flat folded state would be the more technical term. So this is an example of what's called an origami tessellation. But in general, what you're allowed to do, the red lines here mean that you fold as a mountain by 180 degrees and the blue lines mean you fold a valley 180 degrees. So they're specifying the relative orientation. But in flat-folding, you're only allowed to go plus or minus 180. And then, it has to exist.

Now, the rules for paper are twofold. One is that it doesn't stretch. It's an isometric mapping of the piece of paper. Each of these little polygons is just rotated, translated, reflected possibly over here. They fit together, so you don't tear the paper. They fit together at all the creases and there are no crossings, because paper can't intersect itself. So the geometry is basically determined if you say, OK, I'm going to view this panel as being fixed. It doesn't move. Then, this one is just going to be a reflection through this line flipped over. And in general, all of these other polygons just have to be reflections, because every time you fold on a line-- and the rules of the game here is you have to fold everywhere there's a crease.

You can figure out this polygon will end up here relative to some original polygon. So you can figure out pretty much what this looks like as an x-ray diagram. And the hard part becomes how are layers stacked. Is this one on top of this one or is this one on top of that one? That's why the problem is NP-complete is that last step, deciding which layers are on top of the other. In general, whenever two layers of paper partially overlap, you have to decide which is on top of the other.

And then, there's some obvious constraints like-- let me get my hands oriented--

this is OK but this is not OK. If there's one crease of paper here and another crease of paper here, then these locally intersect. So this is OK. This is OK. This is not OK. That's the rules. We don't need to know those rules super precisely to understand this proof fairly intuitively because everything's going to be small and quite local.

The basic idea is the following. You take a piece of paper and you fold it along two parallel lines. So fold along this line and then along this line. And this is called a pleat and there are two ways to do it. You could do one mountain, one valley, or you could do the other one mountain and the other one valley. You can't make them both mountains because then, you'd locally intersect yourself. So it could be this way or it could be the other way I just had it. It's a little tedious. And so you have to make that choice. Which one is a mountain? Which one is a valley?

That's going to be our true signal. We're going to consider one option true, the other option false. And as a point of notation, we're going to have an arrow in every one of these-- we call them wire gadgets. In general, wires are how we have truth settings. And so relative to the orientation of this arrow, if it's valley on the left-- blue is valley because I guess rivers are in valleys-- then we'll consider it true. If there's blue on the right, we'll consider it false. Question?

AUDIENCE: Just a question about the SAT. Once you draw all the red and blue lines, there's just one unique answer?

PROFESSOR: No, the answer's not necessarily unique, but there will be a unique yes or no answer. Either it will be flat-foldable or not, but there actually may be many flat foldings consistent with a given mountain-valley assignment in general. So we're not worrying about that too much here. We're just worrying about whether it is feasible, whether there's some ordering that makes things work out.

So the idea is this could be your variable. Some reductions have variable gadgets. Some don't. This one is just going to have a wire. The idea is if you say, well, there's a pleat over here in the paper, then it could be pleated one way or the other. And that serves as a variable, as you make one choice or the other. What we'd like is to take that truth value and duplicate it, make many copies. But that's not the next

gadget.

The next gadget is a not all equal clause gadget. So this is going to be a reduction from not all equal 3SAT. And I have the gadget pre-folded here. So this is what in origami world is called a triangular twist. It's a slightly weird triangular twist, in that this is 35 degrees instead of 30 being the usual one. And so it's just a little bit larger and that causes a problem. So this is an example of a valid flat-folding of that gadget.

And if you look at the three pleats that are coming in, notice the arrows are all pointing in. So it's sort of symmetric. It's going to be one of the valid ones. So I guess it's actually written on the diagram here. This one is considered true because there's a valley on your right, this one is considered false because there's a valley on your-- oh I'm so confused-- on your right, and this one is considered true because there's a valley on your left-- oh, sorry, false because there's a valley on your right. If you're walking down the arrow, then the right-hand side has a valley.

And so this happens to fold because they're not all the same. If you tried to make them all the same, which should be like reversing the true, it's going to look something like this, which is actually easier to see back here. So now, they're all trying to fold the same way. But the problem is on this side. These guys will all come and intersect in the center.

And I think we have a diagram Here And in x-ray, this is what must happen. And there's this little area of intersection where you cannot resolve which of these three things are on top of each other because this one wants be on top of this one who wants to be on top of this one who wants to be on top of this one. So there's a cycle of constraints. No one can be on top of each other in this little region.

Then, in the regular triangle twist, this is a point and everything's fine. But the way these angles are set up, there's this area of overlap. And in that one situation, you're toast So if they're all true-- but of course, everything symmetric. If you flip it over, then it's the same if they're all false. So you forbid all true. You forbid all false. Everything else will work so this is not all equal. This will be flat-foldable locally if and

only if those variable assignments are not all equal.

Now, we need a couple more gadgets for this to work. One is the ability to duplicate a signal and the other is a crossover. So here, this is a proof by Burn and Hayes, by the way. This is, I think, the first or second paper in computational origami. So it got things off to an exciting start. This is a splitter/negation gadget. This is what it folds like. It looks a little bit weird because my paper's a little bit small, but the point is there's essentially only one way to fold this. And if you take 6,849, you'll know this little local analysis that forces different mountains and valleys to be the same.

For example, these two have to be opposite, which forces if this is a true, this one must be a false. So that's your negation. And also from a local analysis of this vertex, if this one is true, this one must be true. And it works and it also works in the inverse setup. That's always going to be true. A natural reason for us to be using not all equal 3SAT here is there's really no preference between true and false. They're not really a logical notion. It's more like red and blue, literally red and blue.

So not all equal is the one that's nice and symmetric. It has no preference between red and blue. And so we're seeing that here. Because everything's invertible, we want that symmetry between red and blue. So that would be one reason to guest not all equal 3SAT is a good choice here, at least for that wire.

So cool, if we have a signal, we can make a copy. We also get two copies. One of them happens to be negated with this particular choice of arrow orientation. If we oriented the arrow the other way, you could call that a copy, but we don't want to take a wire that's pointing backwards because we need to attach this to something else. These all have very particular angles, but it turns out that will be OK.

One more gadget is a crossover gadget. This is very simple. It is just one-- this is not the pre-folded one-- you do one pleat and then you do the other pleat. And you can see from the crease pattern, you have to do this pleat and then do this pleat. But it's completely independent. I could do each pleat either way. It doesn't affect the other one. So that's crossover, done, easy. This works with pretty much any angles.

And then, it's just a matter of checking that you can fit all the gadgets together. This is something we did in our book to get a picture of how everything works. But in general, the idea is on the left side of the page, we have all of our variables. Those are just wires so they can go one way or the other. So the idea is here is x2. I just want to bend it so it's pointing downwards so I'm going to use the splitter/negation gadget. So we get this copy of x2 just going off. It's going to go up to the top of the page, no big deal. Just throw it away. Then, we're going to go over here. Something happens to cross us, but we don't care.

Now, what I would like to do-- looks like this clause involves x2. So I'm going to do this funny kind of turnaround thingy which will end up making a copy of this variable pointing straight up and also will make a copy of the variable pointing in this direction. And so in general, I'm going to keep doing that. And whenever I need a copy, I'll just use an appropriate parity of turns to get a positive copy.

So this is negated and then this is a copy of the negation and then we flip it one, two, three times. So in the end, it should be positive. So this is the tricky part to check. Here, we have a not all equal 3SAT clause where three things come together. And that's going to be foldable if and only if the three variables that we're combining have a reasonable assignment. And then, we just have to copy all these things and get them to meet all at those nexuses at the top.

So definitely tricky to make sure this works to guarantee all the coordinates can be encoded with a polynomial number of bits. And therefore, this is a strongly NP-hard proof. But I'm going to wave my hands of those details. Questions? If it's any consolation, the original proof also waves its hand at these details.

AUDIENCE: If you have a fold pattern like this, can you tell what order you would need to do the folds in?

PROFESSOR: There's no such thing as order of the folds here. All of the folds would be folded simultaneously, more or less.

AUDIENCE: Instantaneously.

PROFESSOR: Instantaneously, to make it even more precise. So here, it was really a question of whether there is a flat-folded state. We're not worried about the motion to get there. That's another topic which you should take 8.49 if you're curious about. But these kinds of patterns, you can't do one thing because there's no pleat that goes all the way. And that's life, unlike the folding problem we saw two classes ago, the map folding where we were doing one and then the other. In the notion of simple folds, there's an ordering, but in general origami, there is no ordering. Cool. Good, that is origami stuff.

The next topic is a more graph theoretic basic question, but it will also relate to a puzzle. And this is the idea of vertex disjoint paths. This is in Jason Lynch's early work, before he was born, no relation. 1975, so early days of NP-completeness-- this problem, vertex disjoint paths in a graph is proved NP-complete.

So the problem is, I give you a bunch of terminal pairs, v1, v1 prime, v2, v2 prime, and so on, c1, c1 prime, a bunch of pairs of vertices. And I want to connect the pairs by paths. So I want to find a path from v1 to v1 prime and so on so that those paths are all vertex disjoint. Here's the hardness proof. Super simple-- this is one of the easiest 3SAT reductions. This is not the end of their paper. They do more interesting things, which I will get to, but first, let's understand this one.

So this is a graph and I've drawn it in a funny way. These are vertices. These are vertices. These are not vertices. Those are just in the plane, they happen to cross, but there's no vertex there. So it's just a straight path all the way through. Kind of similar to what we did with phutball-- the idea is that from v1 to v1 prime, there are two paths, at least if you stay here. You could try to go into a clause path and wreak havoc, but I think you will be doomed, I hope.

So the idea is v1, maybe you choose the blue path. That will prevent c3 from using that path because there's an intersection here. It won't prevent anybody else because there's no other vertices here. So I use the blue path, c3 cannot use that path. If I use the red path, c1 can't use this one. c2 can't use this one. So again, we are blocking the thing that it would be setting false. So that means, let's say red-- I

forget in this picture-- let's say red is true. Then, c1 has v1 bar in it. c1 is happy if you choose the blue path but it's not happy if you choose the red path.

Now, c1 to c1 prime only has to be happy if one of these three things are happy and that's 3SAT. At least one of the things should be set correctly. Then, there is a path from c1 to c1 prime. Otherwise, there'll be vertex intersections. You won't have vertex disjoint paths. So in a graph, done.

Now, let's make it a planar graph, where things get fun. So this is the original gadget. I've stared at it for many hours, but it's actually pretty simple after you stare at it for enough hours. Let me try to convince you of that.

So I'm going to follow the same kind of outline here, but we have to get rid of these. These are fine. Those are intersections. But these crossovers, we need to build a crossover gadget. So the idea is instead of v1 to v1 prime being a single path-- that's going to be a problem. Instead of that, I'm going to have it be several paths and they're going to be nested brackets. Chalk-- this is the kind of thing we're imagining. We have some vertex pairs which can connect like this.

And there are two ways to connect them in what will end up being the diagram. We could connect them like this. Notice I'm still connecting the same pairs if I do it right. And the whites do not intersect each other. They're vertex disjoint. The reds do not intersect each other. They're vertex disjoint. But if you tried to switch from white to red, you'd get intersections. There's exactly two settings here. So this is a wire. Once I choose over here white or red, all the others are forced to alternate.

Oh, sorry. Once I choose up or down, the rest are forced to alternate up or down. If I choose white here, they're all white. If I choose red, they're all red. This is independent for each variable, so independent for each wire. So that's what's going on here and if you look closely at the bold lines, you've got a bracket here from v1 to x1. Then, you've got a bracket here from x2 to x3, then a bracket here from x4. But in general, there would be lots of those brackets going all the way down the picture.

And the cool thing about this is you get the ability to crossover, because now if I just restrict this picture-- it's hard to see-- if I just restrict to the white part, you can go from top to bottom. You just need an appropriate kind of zigzag. That will let you go from up here to down there without intersections and that's what's happening with this line. It's going over, around the brackets, and down. And so the c1 to c1 prime can remain a single path. It can always get through.

If you have this gadget, then you can have an intersection with no trouble. But v1 is still communicating information down the line of one setting or the other, 0 or 1. There's some details to check there, but that's the idea. And that's the end of the Lynch paper.

Now, we were looking at this earlier this year and we can prove an even stronger result, which is what you might call planar vertex disjoint paths in a rectangle. So the idea is you have a rectangle of one-by-one squares. Each one-by-one square could be a terminal and a terminal pair or it could just be a blank space so you can route paths through. And my goal is to find vertex disjoint paths connecting all the terminal pairs. But furthermore, I want every square of the grid to be occupied by a path.

So it's not quite Hamiltonian path. It's like you have a bunch of paths that collectively fill the entire grid. Why? It's motivated by a puzzle, but it's natural enough. This is a more specialized version. So this is also NP-hard and we're going to mimic the same proof. And this, for example, is a gadget to do, say, this little picture where you have two choices. So maybe the v1 dot is actually here and the idea is you could either follow this path or you can follow this path.

And what's also drawn here are lots of other terminal pairs. These are what you might call adjacent terminal pairs, what you might also call obstacles. Because the paths have to be vertex disjoint, no path can go through a terminal. Any of these dots serve as obstacles so we effectively simulate this blank space that you're not allowed to touch.

Now, the fun part is maybe those terminal pairs are connected by the single edge that joins them and doesn't get in the way. But there are other possibilities drawn

with these blue lines. So instead of going here, I could do this. Instead of going here, I could do this and this. And then, if you look at this picture, every pixel has a path through it in this case. And then, here's the other case again and now I'll fill these two pixels with this pair. And now, every pixel has a path through it. That's my goal is to make sure every pixel has a path through it.

So we're just trying to preserve all of the solutions we have here but with the additional constraint that all of the-- because we already know this is simulating 3SAT. We already know it's hard. We just now want to have this extra constraint that every pixel has a path through it and that we live in a rectangle.

So this looks good. As you might imagine, there's a little bit of finesse to make these gadgets work. You notice some of these guys are oriented horizontally. Some of them are oriented vertically. That's to make it work, but this is easy. What you're about to see is less easy.

So let's do the crossover gadget, because that's the heart of everything. And when we were starting on this-- I think this was in May-- I was very happy when I could finally draw this picture. Sorry, this is actually the final gadget which works but you can see the same kind of picture. I've got a bracket here that corresponds to this bracket here and there's another bracket here. I've just rounded it to integer coordinates and made sure that everything else can be filled in. Easy, right?

In the sake of education, let me show you all the things that went wrong in this proof. Because it was only a few months ago, I mostly remember everything that went wrong, which was a lot. So this was an early version of the gadget, not actually the first one but it serves its purpose. So here, we have the two settings. It's corresponding to these two settings, the up and the down setting, the true and the false setting. We've got this path coming through in both cases. It takes a slightly different path over here. The brackets are flipped.

So the first thing I want to point out are these circled nodes. The obvious way to draw this picture is with all of the terminals in the center row. That would be reasonable. They're not drawn that way here because imagine at this point, we're

down here and we have this big path over here. Then, there'll be three pixels here which somehow have to be filled. And that is trouble because three is odd. So this is what you might call issue one.

Wherever you have empty space, any group should have even area because if you look at two guys that are adjacent and you replace them with some other path, together, that forms a polygon. And in the square grid every polygon has even area. So you will never fill exactly three spaces. So the major revolution which I spent many hours figuring out is if I just move these points up one spot, everything just works. I was so happy. So I figured, OK, proof's done. I drew these two figures and forgot about it.

Then, a couple months later, we want to write the paper and then we get to issue two. So issue two is well, this is great. This works fine if the clause path is present. If the clause chooses this vertical path, this will fill everything. This will fill everything. But what if the clause path is absent? Clause has three choices. It may not choose this vertical path. So then, somehow, this stuff has to be covered using all the extra filler stuff.

And here, I couldn't quite figure out how to fill these things. Well, maybe I should add a couple more rows in the middle. That didn't seem to help. I thought this one was OK. It looks fine until I notice that well, either I connect these two dots by a length two path or I connect it by this path. But in that case, this pixel is uncovered by paths. And this is bad, because there's exactly one such pixel, which means I have another parity problem again.

So I think what we ended up doing-- and when you're really doing it, you don't immediately realize it's a parity problem. I tried trying to fix this many, many times and every single time, I had one pixel uncovered. Oh, bet there's a theorem here. And then, I proved, oh, yeah, it's got to change parity. So what we ended up doing was adding a row down here. Make this path slightly longer. That adds exactly one uncovered pixel so it should fix the parity problem.

Many hours later, we came up with this. So this is good. Every pixel is covered in all

four cases. The bottom row is that the clause path is in and the top row is the clause path is out. And as you can see, lots of careful orientations of these little terminal pairs to make sure in all cases, you can cover everything. This was frustrating because you'd fix one and then you switch to the other layer and say, oh, I broke this other one. I rotated these guys and now, it's not possible to do this, so you better-- and it would be like, you'd rotate this and then rotate that and rotate this. And luckily, no bad cycles happened. You could always resolve it.

And so finally, we got this picture, except for issue three, which is another parity problem. I was blown away. It was like every day, there was another parity problem. Of course, the deadline is tomorrow at this point. Luckily, I started three days ahead so there were exactly three issues.

So remember this gadget, which was the choice. I didn't say this is a variable choice. You have two choices. If you want a clause choice, you connect two of these gadgets together and then you have three different paths you might follow.

If you stare at this enough-- and I just wanted to make sure, OK, can I plug these together? I thought, OK, I'll just copy and paste and lay things out and I had a parity problem. I was like, oh, man, what's going on? Well, if you look here, this is at one column and this is a column four positions over. So in particular, the two choices here have the same parity. I think it's mod 2 not mod 4 but I could double-check. And you just cannot build this if you want these things to be an odd distance apart because parity.

So then, you look at this gadget and you realize it has an odd number of columns. We've made it an even number of rows when we added that last row but now it has an odd number of columns, which is kind of weird but that's life. So we need to make it even. And so I think if you focus right here, we add another column. It's now right here so where there used to be two, now there are three. We have this little trick to fill things in and not too much changed.

I think I also cleaned up the gadget or removed a couple extra columns on the right, but the key thing was to add a new column that fixed the parity. Now, the number of

columns is even and now, I have to prove this. But it's pretty easy to check. You can combine these gadgets together and they match up. And then, if you have blank space, it's also even by even and so it always has a perfect matching and so you can fill in any blank space with extra pairs.

Those are the gadgets except there's one more issue I haven't mentioned.

AUDIENCE: Yeah, there's a parity problem with your parity problems. There's only an odd number of them.

PROFESSOR: Only an odd number of parity problems-- luckily, the next problem is not a parity problem or unluckily-- I don't know. And this wasn't really a problem. I knew it was going to happen. I just had to draw the figure but it's good for an exercise. Think about what could possibly be missing. We spent all this time doing crossover gadgets but there's also the case where the wires cross and we actually want the vertical wire to block the horizontal wire in a particular choice.

This is actually really easy to do. So here's the gadget I was just showing you without all the stuff filled in. Those are just the terminal pairs. And this guy goes to the left. This guy goes to here. This guy goes to the right. If you just add in an extra pair here or an extra pair here, you will block blue or red, correspondingly. So if we go to the previous slide here, blue goes through this position but red does not. And down here, blue does not go through this position but red does. And so if you add in this extra little pair, you will force red or blue at that intersection.

And that was the little squares in the overall diagram. So if you choose this vertical path, you force the horizontal path to be the other thing. That's the proof. I hope there are no more issues but it was exciting. So just to give you a flavor for parity issues, there are all over the place, especially when you're on a grid. Be careful of them. That's the life lesson.

I'll tell you a little bit about where this kind of problem comes from. The earliest reference we know is by Sam Loyd, a famous puzzle designer. In 1897, he posed this puzzle, an incredibly complicated puzzle, lots of stuff in it. And then, I think the

next issue, he published the solution, which is this. Hopefully, I remember the problem. So you have these houses and you'll notice each house faces an exit.

And what you'd like is for each house to be connected by a pathway to the exit. But if you just did that, they would intersect each other. So you want vertex disjoint paths in the grid from the houses to the exit that they face. So that is a terminal pair vertex disjoint path problem. Now, he didn't specify that those paths should fill the entire grid but they almost do. Other than this little corner, they do and a few little corners at the extremes. And you can fix it. Instead of going like this, you can go zig, zig, zig, zig, and zig, zig, zig, zig, zig and make it into a solution to the problem I just said.

So that's 1897. In modern terms, this is known as the Numberlink puzzle. Nikoli is a famous Japanese puzzle publisher. These are two Nikoli books that we got in our last trip to Japan a couple months ago. These are both for the Numberlink puzzle. So you turn to an arbitrary page and you have some picture like this, which here's an example. You have pairs. You have a bunch of numbers and blank spaces. The idea is that there's exactly two of each number. Those are your terminal pairs. You want to find vertex disjoint paths that connect them and in the back, there's solutions if you can't solve one.

They have easy ones, medium ones, hard ones. So how hard is it? NP-hard because this is exactly the problem we were just solving, if you want to find vertex disjoint paths in the grid. There is this issue of whether you are specified as one of the constraints that you must visit every square. But do a quick visual scan here-- in every solution that I see, every square happens to be filled.

Now, that might be the way that they designed the puzzles, the only way to solve them is for every square to be filled or it might be that's a constraint to the puzzle. And different write-ups suggest different ways. But what we proved is, both ways, it's hard. Whether you give that as a specification or you make it possible to do, it is NP-hard because we started from a reduction that didn't have to fill every pixel and we turned it into one that could fill every pixel if you wanted. Yeah?

AUDIENCE: There's a sense in which these puzzles are sparser, in terms of the number of vertices than your reduction. Is it possible that you could define it in such a way that these are unsolvable?

PROFESSOR: It's an interesting question. So our reduction has most of the cells filled with numbers. These you could say, if you have an n by n grid, maybe these only have order n numbers, whereas we have order n squared numbers. That would be interesting. I don't know the complexity. The one thing that's known is for one terminal pair, it's polynomial time. I think even two terminal pairs is an open problem. So you need to solve that before you could solve n terminal pairs for an n by n. So I don't know how many people worked on two terminal pairs. I've thought about it a little bit. It seems very difficult.

Anyway, a little more to this story. In fact, we first encountered Numberlink in the form of an Android game called Flow Free, which some of you may have played. It became popular for a few months there but it is exactly Numberlink as I have defined it. And once you know that, then there's also Number Link and a zillion other clones. And I don't think Nikoli publishes any of these. But it's all from 1897, so unclear what the copyright is here. But in particular, Number Link I like. It has the nicest GUI.

But in the beginning tutorial, it says, well, there's actually two modes you can play. There's zigzag mode, where you can draw paths however you want. That's what I've been talking about because graphed theoretically, that's the natural thing. Then, there's this other modes called classic mode where if you draw a complicated path, it gets-- "autoshrinked" is the general term.

Now, again, it's hard to have an authority here in what are the rules of Number Link? But one definition is you consider all possible paths between these pairs that have the same topology. You can try to go this way or you could try to go this way. Here, you have these two guys on your right-hand side. Here, you have them on the left-hand side. In general, that's called a homotopy type of that path.

Consider all paths with the same homotopy type. Your path should be shortest

among all such paths. That is one notion of autoshrinked. I don't know if that's the official one, but it is a--

AUDIENCE: That has to do with the order in which you set the things.

PROFESSOR: This description suggests a particular order. So you'd also maybe need a proper ordering. Yeah?

AUDIENCE: So the 1 to the 1 here, there's multiple ways-- well, the other way, there's multiple ways to get through that square grid, I think. You can go down and all the way around or you could go--

PROFESSOR: Right. These would both be the same length. And I think those are both considered valid, although you'd have to play in this particular--

AUDIENCE: So it wouldn't change either of those.It would take the one you chose.

PROFESSOR: Right. So you still have some flexibility. There are multiple shortest paths in this world, because we're Manhattan metric.

So this raises the issue of classic Number Link. And most of the puzzles-- I think all the puzzles-- I haven't checked every single one-- actually follow these rules. But again, I'm not clear what the rules actually are, whether they require this or this. But luckily, there's another paper the covers that case. It's in Japanese but it considers the classic mode, where you have to do some kind of locally shortest paths. And the gadgets critically exploit that feature.

It's a reduction from planar 3SAT. We might look at it when we get to planar 3SAT, which is the next class or two. Or not-- there's a ton of planar 3SAT proofs. But anyway, that's been covered already. The actual order of events is we proved our result and then we're like, uh oh, this is Numberlink. Uh oh, Numberlink is NP-complete and that was known a few years ago. Oh, good thing it's a different game that we studied. So there are two versions of Numberlink now. They're both hard. So that's the good news and that's all for today.

This is the *Super Mario Brothers* edition of 6.890.