ERIK DEMAINE: Welcome back to 6.890. Today will be our last lecture about pure 3SAT, so to speak, and we'll be focusing in particular on circuit SAT today.

But before we get started with more reductions, I want to highlight two styles of 3SAT proofs that we've seen. This is orthogonal to the circuit perspective versus logic perspective. And I think it's interesting to highlight the two different approaches.

So approach one I'm going to call dual rail logic. These are not the usual names. We're making up names here. There aren't really names for these types of reductions. And the other type I'm going to call binary logic. These names come from actual circuit stuff, real hardware.

So for example, all the Nintendo proofs we saw are in the style of what I would call dual rail logic because the variables have two outputs. And the idea is that this one is activated when x is set to true. This one is activated when not x is set to true or x is set to false. Never should both of them be activated.

So you have a separate rail, separate line for the true case, and a separate line for the false case. And this is usually called the variable gadget that does that. And I'm going to make up some more terminology. I'll call this semi-wires to distinguish from wires.

So over here in the binary logic case, this is sort of a half of a distinction. It's a semi-distinction. This is my attempt to unify things slightly.

So the idea is that these semi-wires, either they get activated, or they don't get activated at all. In the unactivated case, nothing happens to them. So they're not really interesting.

In the binary logic case, the idea is that the wire gadget is active in all situations, but it can be active in two different ways. You can either set it to the true state or you can set it to the false state, and it represents both sort of in one wire.

Now, this distinction is going to get a little bit of egg. But we saw an example of this kind of proof when we were talking about crease pattern, flat foldability, and p hardness. The wire was the pleat, and the pleat could be folded one way or the other. But either way, the information is right here, and the same wire gets put into this gadget whether you're true or false.

Whereas over here you're connecting to this clause only the true half wire or semi-wire goes to this clause. The false one goes to this clause and to this clause. So there's a distinction between here where the wire's been split in half, and here where it's all in one place, and the wire goes to the gadget whether it's a true or false.

I think that's what I wanted to say. My main intent here is to highlight what kind of gadgets you need. So we have a wire gadget on this side, a variable gadget on this side. For binary logic you also need a splitter or a split gadget to make copies of the wire.

So the idea is you have one wire coming in, there's some magic split gadget which makes two wires or three wires, whatever, coming out. So then you can repeat split however many times that variable occurs. You can make a copy. In the non-monotone versions of 3SAT you may also need a negation or not gadget. If you need negation, then you'll need a negation gadget.

In some cases you also need a terminator gadget in this perspective. A terminator gadget is just the end of a wire, you might say. So if you want to just not use a wire, then you would put a terminator gadget. And the idea of the terminator gadget is it leaves it free, leaves the wire free to choose true or false.

Now, sometimes you can get away without a terminator gadget. For example, if each of your variables occurs exactly four times, then you can just use two splitters, make four copies, and then connect those directly to the clauses, maybe with some negations. So in that case, you wouldn't need a terminator.

Over here we're using a terminator in a certain sense, which is the edge of the paper. If a wire goes to the edge of the paper it becomes free. So that's sort of the terminator gadget here. Though we didn't make explicit, it was part of this overall plan.

In both cases, in one and two, you probably need something like a turn gadget and maybe a crossover gadget. We've seen these in various forms. In this case, the turn gadget was very explicit. I mean it was also a splitter and also in negator, but in particular, it changed the angle, and we needed that in order to take this truth value and send it up to clause and aim everything right.

And in general, if you were doing any kind of embedding into two dimensions, you're probably going to need a turn gadget so you can move your wires or semi-wires around.

A lot of the time people might call this also a wire, so that's why. But I think it's useful to have a word to distinguish these kinds of wires which are only active or inactive, versus the wires which are always active but in two possible states. Hopefully that clarifies or makes more explicit the types of proofs we've seen so far.

Now, today we're going to do a couple of circuit SAT proofs, which all of this applies to circuit SAT proofs, as well as SAT and 1 and 3SAT and that'll equals SAT, that'll equal 3SAT. Circuit SAT you're going to have-- I guess in principle you could do it dual rail logic or binary logic, although usually we probably think of them in this capacity in binary logic.

And you're going to have wires. You still need splits because you need to be able to take the output of a gate and send it to multiple other gates. In addition to a not gadget, you're going to want some other universal logic gate. So we're going to talk about that in a bit after we see some examples.

Our first example is called Akari or Light Up. This is another Nikoli puzzle like Sudoku and Numberlink we saw last time. So here's a sample input and a sample output. Ignore the black dots there.

So you're given a grid. There's blank squares and there's obstacle squares where you're not allowed to put anything. Some of the obstacle squares may have a number between 0 and 4 on them. And that number represents how many, among the four neighbors, edge neighbors of that square, how many of them should have a light? These blue circles are lights.

OK, so like this one has zero lights in those four positions, this one has two, this one has four, that's forced, and so on. A lot of the cells are free, but whether you're lit or not.

And then the rules are so you can think of a light as a rook in chess, so it can see everything horizontally in the same row, up to obstacles, and in the same column up to obstacles. So this guy, for example, lights this portion and this portion. This guy lights that portion, and the thing here, and so on.

And the goal is, first of all, to light the entire board, so that's why it's all green, it's all lit in that way. And no two lights should like each other, because then it's too bright I guess. I don't know. So no two rooks can attack each other would be another perspective. I think those are the rules.

So you've got to satisfy the number constraint. You can only place lights in the blank squares. You have to light everything, and they can't see each other. So this is a valid solution. This is puzzle one on the Nikoli website, and there's a ton more if you want to play.

AUDIENCE: So you're saying they can't be in the same NOR column?

PROFESSOR: I mean they can be in the same column, but they have to be separated by an obstacle. So they can't see each other within a sort of sub-column or within a sub-row as given by the obstacles. Cool.

So that's the problem, and now we're going to prove that it's NP-complete. This is the nice proof because it's relatively simple and it illustrates a kind of circuit SAT and style reduction.

So for starters, and I would say most SAT proofs, a good starting point is a wire gadget. You want to think about can I make some construction that has ideally exactly two solutions, one to represent true, one to represent false.

So here are a variety of ways to do wires. And this is a fairly flexible kind of game. The basic one looks like this. Most of things are going to be obstacles to tie it down, make it easy to separate gadgets. But here we have two blanks squares, a one, two blank squares, a one, two blank squares. That's sort of a simple kind of wire because you know from this one, either x or x prime is a light. Exactly one of them is. So that forces alternation. Done.

Why? Because if we choose x, we can't choose x prime because then they're in the same sub-row and all the way down the chain. So in general, if x is true, the idea is that the x's are the lights. If x is false, then the x primes are the lights.

Now, that's a basic wire. But part of the wires we need a turn gadget. So this is a 90 degree turn. Just check that this is still lit. If these guys could not see each other, I think this is black. It's also OK. The alternation's forced by the one. But some flexibility.

And let's see. The other fun thing you could do is spread out the x and x prime. If you put a bunch of zeroes here to force that there's no lights here, because this has to be lit again, one of these has to be active, and the ones force an alternation.

So this is helpful because if our wires just look like this, and this will come up again today, then you're forced to be congruent to zero mod three. First your width has to be a multiple of 3.

And this can be an issue. If you have a bunch of gadgets each of the different sizes- - maybe one is 5 by 5, another 3 by 3, and other fun prime numbers, then you might have trouble actually getting gadgets to meet edge on. Maybe there's a gap of one or two or something between the gadgets.

But if you have a set up where you can adjust the wire lengths to be any integer, that's good. You can do this most of the time, but then when you're getting close to the gate, if you're off by one or two or three, you just make it a little bit longer and you're all set.

AUDIENCE: The unfilled one can also just have a crossover gadget then?

PROFESSOR: Ah, sorry?

AUDIENCE: Since you have nothing-- it's very natural to have a crossover gadget by putting two of them orthongonal.

PROFESSOR: Oh yeah. You can make a crossover gadget easy in this set up. I think that is a future slide.

So we'll do crossover in a bit. Let's start with-- yeah, you can very easily do crossover. We won't need a crossover here. That's what I mean.

So we definitely need splitting, though. In general, whenever you have wires you've got to be able to make copies, because if your variable only appears in two places, 3SAT is easy, our circuit SAT is probably easy.

So here is a splitter. This is the main mechanism. I guess these ones and this two. So these ones force an alternation between here and here. These have to be opposites of each other. And then the two-- so if both of the x's are present, if this is absent, then these two have to be present by the ones. And so by this two, that's all there is, and therefore this is absent. So these guys have to be the same. These guys have to be opposite.

And conversely, if this is present by these ones, these have to be absent, and by this two, this has to be present. So this is almost sort of a-- well, you could also put some ones here I guess if you felt like it.

Anyway, if you think about the parity here, when we're represent a signal coming into a gadget, there's x on the left and x prime on the right. So that means coming out here there's not x prime on the left and x on the right, so this actually a negated copy of x. And up here with some turns, we have a regular copy of x and a regular copy of x.

This is actually a fairly common thing that happens. When you try to split maybe you also get a negated copy, or instead you get a negated copy. This is actually great because it kills two birds with one stone. You could use this as a splitter. It kind of makes 3D copies of your signal. But you could also use it just as a negation gadget, as a not gate.

I think I have a slide of those done explicitly. Yeah. So here we're effectively using a terminator gadget. We're saying, well, we could throw away this negated copy because we already have two positive copies and that's enough for a splitter. So if we just end this wire here, great, we've got a split. Or if we throw away the two copies of x and we just want the negated copy of x, then ends up looking like this.

So we get a split gadget and a not gate essentially for free out of this. You could just present this gadget and say, oh, I also have terminators, and then you know that you get these. But for fun, he drew them and I'll show you them. I won't show you them in future proofs. Cool.

So let's see, what do we have? We've got a wire. We've got a split. We've got a not. We've got a terminator. We need some kind of clause or some kind of logic gate. And then possibly we also need a crossover. We have turns.

Next up is a gate. It's sort of a mega gate. I'm guessing McPhail tried various little configurations and then found one that did interesting things. It actually does two interesting things at once.

So we have x coming in on the left and y coming in on the right, so both of them have this x prime and y prime, then negated copies. And then we've got some fun stuff in here. So let's start with maybe the bottom, which is supposed to be an OR of x and y.

So let's say that they are both false, for example. And in general, we're going to have to check all the cases, but I won't do all of them. So let's say this is absent. This is absent, meaning the it's lit from the other end.

AUDIENCE: If it's false it means that x prime is present.

ERIK DEMAINE: Oh, right. Sorry. If they're both false-- I though it was good-- then both of these are present, and therefore, we're going to get hopefully that this one's absent because the OR of false and false is false. So the claim would be that A prime and B prime must be activated.

AUDIENCE: If they weren't then you couldn't light both A and B.

ERIK DEMAINE: Right. These are at best OR'd together. At most, one of these is present. These regions both need to be lit. So we're going to need at least one of them to be on, but once one of them is on, we know that z is absent, and then the other one must be on. Right, this too. Good.

And then we should up here get x NOR. So when these were both false and present, these were present, these are absent, and therefore, this is present. And the x OR of zero and zero is zero. The x NOR is 1, so that's good.

So in general, you check all the cases and confirm this is the case. Again, by putting terminators on here, we can get either x OR y, or x NOR-- x NOR y-- it sounds funny. That's great.

This is the one that we really care about. The OR gate-- well, so different answers to this. But you should know, and we'll talk more about this later, NAND-- or let's say NAND gate is universal. And a NOR gate by itself is universal. They're just good things you should know. They go back at least 50 years.

AUDIENCE: What does universal mean?

ERIK DEMAINE: So universal means that from those-- if you're just given a bunch of NANDs, then you can construct any logical get you want. I will talk about that more in a moment.

So this OR is particularly interesting because we already have NOT, and so from OR and NOT we can make NOR-- it's just the NOT of the OR. And so from that, we should be able to make anything. Which is great, except for one thing which is the crossover gadget.

So in the case of Light Up, crossover gadget is easy to construct. But because we have x OR anyway, we can use this fun fact that-- this is the symbol for x OR. If you have x OR gadgets, you can make a crossover gadget for free. So this communicates x to x, y to y. There's four cases. I won't check all of them.

But that's cool because we have x NOR, we can negate it and get x OR. And then we can take three of them, plugging them together, and we get a rather complicated crossover gadget if I expanded it out.

There is a simpler one, but we don't care about simplicity here. These are just constants, and we just care about polynomial. So that's Light Up as empty heart.

AUDIENCE: Do we need crossover gadgets in general for circuit SAT? Is planner circuit SAT hard?

ERIK DEMAINE: Please wait. Yeah. This is something-- well, yeah. I want to talk about that after the next example.

AUDIENCE: OK.

ERIK DEMAINE: Which we'll get to an answer to that question. Which as far as I know hasn't been explicitly raised before, but the answer is basically, you don't need crossover gadgets, and for all the cases I can think.

So that was Light Up is NP hard. It's also in NP because a certificate is just where do you put the lights. There's only polynomial number of places to put them, and checking it is easy. Cool. So NP-complete.

Oh, here's for fun an overall construction, which McPhail worked out. So putting all the pieces together and possibly using-- I think here there's maybe no crossovers.

So in general, with Circuit SAT you can construct sort of any Boolean formula you want once you have a universal set of gates. But yeah, you can work through details here.

It's easy to convert between AND and OR just by negating the inputs by De Morgan. So you've got x OR y is NOT x and NOT y or vice versa. So as long as you've got NOT an OR you can construct AND and vice versa by negating things appropriately. So that's what he's doing here for getting some ORs and so on.

And because there's two copies of x-- well, x is sort of getting split here. I guess you didn't have to split x. You could just connect directly. This is a pretty nice clean construction. It could actually make decent resize puzzles with it.

AUDIENCE: So you end up with the wire carrying the formula just by doing all these combinations?

ERIK DEMAINE: Right.

AUDIENCE: And then what do you do?

ERIK DEMAINE: Ah, good point. Thank you. There was one more point to this picture, which is this zero. This should be an OR of x false, and this OR or this AND and blah, blah, blah, which is this thing.

AUDIENCE: So if that wire's carrying true, then you don't need to put a light next to the zero, otherwise you would have to, and that means the puzzle can't be solved.

ERIK DEMAINE: Right. So for this puzzle to be solvable, we better not put a light here, which means it has to be here, which corresponds to this formula being true, I think, if I got that parity right. Or either way. You could put a one there and force there to be a light.

So this makes the puzzle feasible if and only if this formula's feasible. Because the wires are unconstrained, except by setting this thing equal to one.

So this is a special kind of terminator, you might say. A true terminator. You want to force the thing to be true in the end. That's usually pretty easy to do. Terminator's usually one of the simplest gadgets to worry about. But yeah, you do need it.

For circuit SAT you need-- let's add that. Circuit SAT you need let's say a true terminator. There are other ways you could do it. Cool. Other questions about Akari? All right.

So our next topic is Mine Sweeper. This is from our poster. We did Super Mario Brothers, and Bruce Lee did Rush Hour, so Mine Sweeper's the last one on the poster. Let's prove it hard.

We're actually going to cover two proofs about Mine Sweeper, because there are sort of two natural problems you might ask about it. The first problem is consistency. So this is I give you a set up, I'm going to give you a partially solved board that you're not given the x levels. So just a regular instance of Mine Sweeper.

Maybe you're told where some of the bombs are. It won't actually matter, because all the bombs that I'll draw here are drivable from this-- I should mention the rules of Mine Sweeper just in case you haven't played. Anyone not played Mine Sweeper? I'm not willing to admit it.

The numbers here representing, among all your vertex neighbors, all eight neighbors, how many of them are bombs. You don't want to click on the bombs because then you die. So your goal is from this partial information, these blank squares represent zeros. From this partial information, you want to figure out where the bombs must be.

And here's a very familiar looking wire. We can't just have obstacles that are wild cards. So we have to put numbers if we don't want there to be bombs there. That makes the gadgets quite a bit trickier to work with.

But this looks pretty similar. We've got these ones, which force an alternation. Exactly one of these two is a bomb, so if x is present, x prime is absent and vice versa. And then this one forces-- I guess actually this one forces these to be opposite. So that forces the alternation down the line. So again, there are exactly two solutions to this wire gadget for where the bombs could be.

And so the Mine Sweeper consistency problem is I give you some picture involving these things, and I want to know is there a solution-- is there a placement of the bombs that satisfies all of these constraints?

Now, you might ask why is that the problem we care about? And one reason if you're playing Mine Sweeper, you'd like to know, for example, could there be a bomb at this position?

So could there be a bomb at this position? I could solve that problem, potentially, if I could solve Mine Sweeper consistency by saying, let's put a bomb there as part of the partial information. Is that consistent with everything else? So add a bomb to the partial information. Is that a consistent configuration?

So if I could solve Mine Sweeper consistency, I could play Mine Sweeper very well by just testing which are bombable positions. If they're not, possible to be a bomb there. Actually click there and get more information and cook until done.

Now, maybe you could solve Mine Sweeper in a different way than using consistency. But one way to solve it would be consistency. Question?

AUDIENCE: By consistency you don't mean unique solution.

ERIK DEMAINE: I do not mean unique solution, right. Yeah. So when you actually want to solve the game, you'd like there to be a unique solution. That's a problem we will get to in a moment.

AUDIENCE: So a blank board, unfilled information is consistent.

ERIK DEMAINE: Yes. Completely blank board is consistent. And it wouldn't be interesting. As you're playing, you imagine you've gathered some information, you add a bomb. You want to say is this consistent. It says yes, like you're in a blank board. And you're like could there be a bomb here. Of course there could be. Then that's not a safe move.

So if you could solve this, you could identify safe moves. Turns out you can't solve it, so it doesn't directly help you. Yeah.

AUDIENCE: So there doesn't always exist a safe move.

ERIK DEMAINE: There doesn't always exist a safe move.

AUDIENCE: What good is it, kind of, to know whether [INAUDIBLE].

ERIK DEMAINE: Yeah. I would say consistency is the wrong problem to solve for Mine Sweeper. But it was the first problem solved. The gadgets are relatively easy, which is going to be fairly complicated. And then we'll get to the right problem for if you wanted to solve the puzzle.

At this point it's an interesting question. You just want to know is it consistent. Did the computer cheat would be one other perspective. Or you could imagine some adversarial Mine Sweeper where you only decide the bombs as the person is playing.

So here, just even telling whether your opponent is cheating is hard. So that's another perspective. Let's finish this proof.

We have a terminator, which is actually a little bit tricky here. You can't just end with ones because that would force this guy to be a bomb. So you end with these threes, which force-- these guys are already bombs because of this picture, and then exactly one of those will still have to be a bomb.

And then we have a splitter. It's a similar picture to Akari. We have this two, which forces exactly to two of these to be present, and these ones force alternation.

Now, I'm orienting these arrows for the splitter and defining true to be when the pointy end of the arrow is present. So that means this is actually a negated copy, and these are unnegated copies. But that's only from the orientation. This gadget's actually symmetric, fourfold.

But if you put these arrows in, which you would do if you were actually following a circuit, their circuits are directed acyclic graphs remember. So this is negation, which is good because it gives us a NOT, it also gives us a split. You can also end these two things and get a turn. So great, three gadgets in one.

So we already have NOT. This is another way to do NOT. And it's interesting here because with Akari we could stretch wires to any length, any integer length that's sufficiently long. Here, we're forced in this modulo three picture, normally. But when we do something like a negation here we get kind of messed up. We're no longer in the same position modulo three. We should be off by one I believe.

So the idea is if you do two negations in a row, then you can end up shifting by exactly two and still have the same signal. So you can see here we have xx bar, but down here we have it shifted two positions over. Or I guess it's more impressive over here, so x bar x.

And normally this would repeat in this kind of pattern. So it actually shifted by one position. Good. That's actually what I want. So see the pattern here, here, here, here, and shift by one. Yeah, or two the other direction.

OK, so that's good. That lets me adjust my wire length. These are also often called shift gadgets. They just let you shift by one.

And whenever you have gadgets that force certain modulo constraints, just having the ability to add one is great because then you can break all switch constraints. You no longer have to live in a mod three grid, whatever, if you don't want to.

So this is important. In general-- I'll add it to the list here. When you're dealing with wires you may need some kind of shift gadget that lets you fix parity issues, or in this case, a mod three parity.

OK, here is a more complicated gadget, which I will not go through. It looks scary. It's not actually a ton going on. It's a lot of gadgets we've already seen. It uses a slight-- I've mentioned a turn gadget. This is another way to do a turn. There's a few redundant gadgets in this paper because it's fun to make gadgets, so I want to have more.

So there's, for example, just a turn gadget here. There's a splitter here. Another turn gadget. This is reflectionally symmetric around the x-axis. There's basically just a turn here, so x is just getting copied and negated into here. z is getting copied and negated into here.

This is the output value, which is getting copied around into here. And then a little bit of magic happens here and here to get these to interact in the right way. And it's essentially just checking cases to see that this computes an AND of these two inputs. But a little bit complicated.

AUDIENCE: Is coming up with these gadgets harder than checking them?

ERIK DEMAINE: Good question. It's unclear. Often when you're designing things, you have a lot more control, and it could potentially be easier to design hard problems than to check them. Because we know it's NP-complete to check this, in a certain sense.

I mean, of course, in this case there's only four cases, so it's not really that hard. But yeah, when you're designing things you have in mind a certain connectivity structure, and then it's a matter of getting things to resolve in the grid and not have things overlapping and so on, which is a different kind of constraint.

I might say it's a more fun one. I would call designing these puzzles meta puzzles, and to me that's more fun. But of course, I'm a theoretical computer scientist. That's why we're here. Anyway, fun to answer that question.

So here's a fun thing. I hadn't seen this paper before by Goldschlager in 1977. They were worried about p completeness, which we haven't talked about yet. But in particular, they gave these two pictures. This one we've seen. If you have an x OR gadget, you can construct a crossover.

Now, we don't have an XOR gadget in this case, but we have an AND gadget or a NAND gadget. This is the funny way of writing NAND, NOT of an AND. And if you have NAND gadgets you can construct an XOR.

So if you plug this picture into each of these three pictures you get a kind of ugly thing, but it's planar and it implements a crossover. Which means if you have NAND, you get planarity for free.

So I'm going to call this planar circuit SAT. It's like catch, this, I think is also a made-up term. I haven't seen it in the literature. Because it's a little bit vague what it could mean. But let's say we're given a planar directed acyclic circuit. And let's say all gates are NAND. Then this is NP-complete.

AUDIENCE: [INAUDIBLE].

ERIK DEMAINE: Ah, right. Sorry. And at the end you want it to be true. Thank you. So you have let's say the sources are unconstrained. So those represent the variables. And there's a sync, which is set to one. And we could say there's just one sync.

So you have some starting points. Those are completely free to choose true or false. And then at the end you have a sync. I guess it just has one thing probably coming into it. Wouldn't make sense otherwise. And we set this to be one. So that's just like satisfying formula, but drawn as a Boolean circuit, and it's planar.

Now, we're going to see more versions of planar SAT next class, but this one as far as I know is newish. It's essentially argued in this paper. Say, hey look-- this paper was not talking about NP-completeness, but the K paper says, hey, we've got these two things, so as long as you have NAND you've got crossovers for free. So you don't have to worry about planarity, which is good news. So no crossover gadget needed.

There is one. It's actually not that hard in Mine Sweeper, but it's nicer when you don't have to do it. Questions?

I guess I did talk about termination. We do need a true terminator here because we're doing circuit SAT. That's really easy. In fact, this would do it. Just ending with all ones means, in fact, if there's zeroes out here, that means this has to be a bomb. And so that sets it to one. So there's a true terminator for Mine Sweeper consistency.

But as we've seen in various ways, Mine Sweeper consistency is probably not what we want to actually play the game. The algorithm I gave you is one way to think about playing the game where you add a thing and you check for consistency.

That's also a little bit weird where you're assuming that the input is consistent, and then you add one bomb and you want to see whether that makes it inconsistent. That's a special case of this instance. Maybe not as hard as the general picture. Everything was consistent up until the moment you added one new bomb information. These pictures don't look exactly like that.

Another thing I want to point out in these gadgets is let's say the number of x and x bars is not the same. Maybe this negation, this equal numbers of x and x bars over here, but there's one extra x bar.

What that means is local to this picture, the number of bombs that get used varies. There's either one, two, three, four bombs if you choose x, or there's five bombs if you choose x bar.

And one of the minor things in Mine Sweeper is that the total number of bombs is given to you as input. So that actually also kind of messes things up. Anyway, luckily, this problem was solved. It was actually independently solved by multiple people. It's claimed in Bob Hearn's PhD thesis, though never got fully written up. And then it got published, so no reason to write it up anymore.

So here is their proof. Both this proof and the previous one appeared in Mathematical Intelligencer.

So they draw the gadgets slightly differently, and so far everything looks about the same. They're not going to use a phase changer in this form at least. So they're going to draw the wire gadget like this, and the idea is that you would copy it, you get things mod three, that will turn out to be OK for them.

But what they really want is that in each diagram there are equal number of x and x bars. Why? And also what are they proving? So this is what they call the Mine Sweeper inference problem. But it's also what you might just call Mine Sweeper.

I want to solve this puzzle. What does it mean to solve the puzzle? Well, it means you're given this partial information. You want to know am I done? Did I solve it? Can I solve it? Can I figure out where all the bombs are? This is like there being a unique solution.

So I want to conclude that, OK, I can figure out where all the bombs are. So can you figure everything out? So again, the reduction is from something like circuit SAT. It's going to be a reduction from circuit unSAT. Unsatisfiability.

So satisfiability you want to set the variables so that the outcome is true. You might say, well, I could try to set the variables so the outcome is false. That's the same problem. That won't change. Just put a NOT at the end.

Unsatisfiability means that you cannot satisfy this formula. There's no way to set the variables to make the output true. There does not exist choices for the x ANDs, such that some f of xi's equals 1. If we do some-- who knows, this is called De Morgan? Probably not.

Another fun fact is NOT there exists, NOT is the same thing as for all or however you want to write it. So if we put this negation over to here, this turns into universal quantifiers. These are extensial quantifiers to for alls. So this is quite a different problem.

3SAT is about do there exist settings for these n variables such that this comes out to be true. Now we're saying no matter how you set the variables, this comes out to be true or false I guess. But again, that doesn't matter. You can just put the negation inside f if you wanted to.

So the key difference here is we switched the quantify direction. This is such a big difference that this problem is not NP-complete, unless NP equals coNP. This problem is coNP complete.

Recall coNP are the problems where you can always given no certificate. Whenever the answer is no, I can give you a short proof that the answer is no. How would I tell you that the answer to this question is no? I would give you a satisfying assignment.

I'd give you xa to xn, where f ends up being 1, then you know this is not true.

It's very hard to prove yes here. I mean you might have to check all possible exponentially many assignments to the xi's. Of course, we don't know whether that's possible. That's coNP completeness. It's the same NP versus p problem, but negated. Cool.

So claim is Mine Sweeper inference is coNP-complete. And in general, the proper statement is Mine Sweeper is coNP-complete. Mine Sweeper consistency is NP-complete, but Mine Sweeper the game, I want to solve the game, is coNP-complete. Cool?

This will be one of the few cases where coNP arises for us. Yeah.

AUDIENCE: Where does coNP lie on this? We don't know?

ERIK DEMAINE: In my one-dimensional diagram, you can think of NP and coNP lying in the same space, but maybe like in a parallel universe. I think. People always draw this picture, NP, coNP. P is here, and this is NP intersect coNP. And some people think that's the same thing as P. Who knows.

So there are parallel universes, one about positive problems, one about negative problems. But in some sense, equal complexity.

In this class we think about reductions as being one call reductions. You take your problem, you map it so that the output when you solve this new problem has exactly the same solution to the original problem.

If you instead think about multi-call reductions where you can make calls to an oracle that solved the target problem multiple times and then do stuff with it-- you could, for example, call that thing, then negate the answer and return that.

In that universe NP and coNP become the same thing. Or NP-completeness and coNP-completeness, which are about reductions, become the same thing. So that's the only thing holding them apart is that we're not allowed to negate the answer.

So I view them as almost the same, but obviously they're different and you have to distinguish one from the other. But in terms of difficulty they're about the same. The difference between NP and coNP versus p space is totally different, or x time, or the other things in my one-dimensional diagram.

Yeah?

AUDIENCE: Would p equals NP imply p equals coNP?

ERIK DEMAINE: If p equals NP, then I'm pretty sure NP equals coNP, because in polynomial time you can do-- yeah. p is closed under a complement, because you can solve the problem then negate.

So I should say NP and coNP are different. But NP-complete and coNP-complete are very close, let's say. There's sort of dual-- for every problem over here, the corresponding problem over here is like the hardest, and they're sort of symmetric or something.

All right. Phew. Back to Mine Sweeper.

So here is the coNP-completeness. We have wires just like before, but I'm drawing them differently. Because part of the issue here is we are told how many bombs there are, and we don't want that information to help us.

So we want every diagram to have an equal number of, say, x and x bars. So if we set it with x or set it with x bar, we use exactly the same number of bombs,

Which means knowing what gadgets are glued together we can just add up how many bombs each one has. That will be the total number of bombs given to the puzzle. And that shouldn't give you any extra information because all of these pictures will have equal number of x's and x bars.

So here we have wire, terminator, and turn. Pretty similar to before, but now checking that everything has equal number of x's and x bars. I think this turn, for example-- no, that one happens to be balanced. Some of these gadgets are not balanced. This one almost certainly.

OK. So in this case, they construct a NOT gate and an OR gate. And then they also build a shifter. This is a different kind of shifter. It still has width and multiple of 3, but we're taking the wire and shifting it perpendicular by one unit. With turns you can simulate the other type of shifter, but this turns out to be the only one that they need.

So let's talk about that first, let's say. So if x is set by this two and this bomb, this one is not set. This four already has three of them, so it effectively becomes a one. So then these two have to be opposites from each. This is sort of where the shift happens.

And then because we have to deal with these bombs that are left over. Let's put some more in. And then this is five, so there's already four here, and so that forces an alternation there. So same idea, but with some bombs added in to fill the space, let's say.

Got to do a one here, and a one here, a one there. This is a problem. I can't put a one here. That would force this guy to be present. So that's probably why that bomb is there, because we don't want to have to say how many bombs are adjacent to this corner.

With regular wires you don't have corners. Good. But when you're doing a shift you have sort of corner. Same thing with the turn gadget that we already saw. OK.

I think NOT is also pretty intelligible, but it's fairly complicated. It essentially involves making another copy of the signal just so we get an equal number of x's and x bars. Because we saw in the previous reduction with a NOT gate, we had this one isolated guy. That would be like this one.

And so if you just have this picture, yeah, it would negate things. And again, this is designed to be mod three, so there's three, six-- that's hard to do-- nine, 12, 15. So things are nicely aligned on the mod three boundary, but they end up with these three dudes. They end up flipping the signal. Here's x, here's also x, which is like the reverse because it's on the left.

But this would have an unequal number of x's and x bars. There's too many x's in this row. So we end up splitting off a negated copy here, basically, to balance things out. And now if you count all the x's and x bars they should be equal. Cool.

And then they build an OR gate. So it's again, looks fairly complicated. Probably pretty simple in essence, but I don't feel like checking all those cases.

What is annoying is that this thing is not mod three aligned. Mod three aligned would be here or here, with the output always in the middle finger. But instead it's here. And so they used the shift gadget over here to offset that by one, and similarly in all three pictures.

And then you get an OR gate with proper alignment. Everything works out in mod three. OK. Cool.

So we have NOT and we have OR, therefore we have NOR, and therefore we can build any logical gate we want. An so we have turns, we have shifts that we need, we have NOT, we have-- did we do split? I think we did split on the previous-- nope, we didn't do split. OK, then let's do split.

So we need to copy our wires. This is a more complicated version of the previous split. Again, with an extra thing thrown in. Like this is not needed. You don't have to put this here, but it adds an extra x bar.

So there's actually two of those things. And so together that offsets all the extra x copies that are on the inside. And so that forces, again, equal number of bombs, whether it's x or x bar chosen. So we can just count how many bombs are needed in this picture. Cool.

This is, again, a symmetric version. So if you're thinking of, let's say, this as the input, and these two as the output, than negation happened. So if you don't want negation to happen, just put a big NOT gate there.

And in this case they give a crossover gadget. In particular because it's not that hard to give a crossover gadget, but it's also not necessary. So you could do crossover explicitly, but in fact, I just looked on Wikipedia earlier today.

We already know that planar circuit SAT with NAND gates is hard. So can we build NAND? And so I looked up in Wikipedia, how you build NAND out of NORs. And the answer is this. And that's planar. So we're done.

You take this picture, you plug it into each of these guys. And you take that picture and you plug it into each of these guys. Make sure to preserve planarity all the way through. It does. And then from-- I should know that's you construct NAND out of NOR. It's exactly from De Morgan law, because this is just negation.

So cool thing is all these constructions preserve planarity which means we can add to our definition of planar circuit SAT. All gates are NAND or all gates are NOR. This is nice because when you're solving next p set or whatever, you can just take planar circuit SAT, if you happen to construct a NAND gate you're done. If you happen to construct an OR gate you're done. No crossover needed.

You'll still need turns, and maybe a shift, and maybe a terminator. You won't need NOT if you build NAND or NOR. You'll need a split, but you don't need a crossover anymore. This is cool. As far as I know a new result, but I mean obviously implicit in all of these things.

Yeah.

AUDIENCE: Do you know if there's any sub-problem, I guess, is that where you don't have universal gates and you can't be planar and get it so hard? So maybe you can--

ERIK DEMAINE: Oh, interesting.

AUDIENCE: You can split wires and you have a variable that has the negation of it coming out, but that's all you can do. No other NOT gates.

ERIK DEMAINE: My guess is that all such problems are polynomially solvable, but maybe that's something we could think about. Certainly plausible that you don't need to be able to construct all Boolean formulas in order to be hard.

But my vague sense/experience playing around with weird other gates, most of them are all equivalent to splits if they're not universal-- something like a split. Maybe with some like five operand operator, you could do something cool.

But usually they degenerate to other split-like things or clause-like things. But if you don't have both, you don't get enough to be hard. We should prove a theorem like that, but I would guess that you need something like this.

Now these are not the only universal gate sets. I think in a moment I will have a-- if you are curious about more of what universal gate sets are, these are usually called functionally complete or functional completeness.

If you look at that on Wikipedia, this is an excerpt from Wikipedia, you see NAND and NOR are the only, among arity two operators, so you have two inputs, one output. NAND and NOR are the only things that by themselves are universal. But once you allow two different gates you can do other things.

So these are particularly interesting. This is false I think. It's called [INAUDIBLE], and this is called top, it's true. And so these are slightly unusual ways for me to think about writing them. Here we have, of course, AND and NOT. But this is right implication, which is like not the left thing or the right thing. We talked about that in the past.

That by itself is not enough, because you can just follow implication chains. But if you also have false, that is universal. So that's kind of funny.

And this is XOR I think. XOR is the same as saying that the two things are different. And this is XNOR. So those by themselves are not universal, but if you have some other kind of one-way implication, then that gives you any logical formulas.

Now, whether these in the planar version are hard, I don't know. But I know that Ds in the planar version are hard.

OK, back to Mine Sweeper. They drew a picture of how everything fits together, so it's maybe nice to have the big picture. But let's in particular think about what the decision question is and make sure-- I mean I claim this is coNP and then we're doing unSATs, but we should actually check that.

At this point we've just simulated a circuit. We haven't thought about what we're trying to solve overall.

Now, we want to know whether we can derive everything. So we compute our formula and then the output is here. And at this point there's just a terminal there saying I don't know what it is. Different from the previous proof. The previous proof said that should be one at the end, and then you have to figure out how to fill in the things.

Now we're saying, I don't know what's there. Can you figure out what's there? And if it's the case that no matter how you set the variables you always get false, then you know what this wire looks like. You know that it's false. And in fact, you can figure out everything. If you know what all the XI's are you can just run through the circuit. So you can figure out the entire diagram.

But in general, just asking the question, can I figure out this square? Is there a bomb or not? In order for there not to be a bomb, you would have to solve this problem. You need to determine that no matter how you set the things you end up with a zero. Or the adjacent position it would be the opposite.

So conversely, what I really care about-- that was the wrong direction. I'm given a circuit unSAT instance. I want to turn it into a Mine Sweeper instance. So I do that, and then the question is-- get this right-- can you derive whether there is a bomb in this square? Can you determine that there must be a bomb in this square, for example? That would be a slightly cleaner version.

And that will be true, if and only if no matter what you do, there's a bomb in that square. Which is like saying no matter what you do, this formula comes out to zero. So any kind of inference question like that is going to require you to solve this problem, or rather, this problem can be reduced to any inference problem like that.

It's easy to get backwards here, so hope I got it all right. And that's coNP completeness. Reductions look the same for coNP. We just start with different problems. Cool. So that was Mine Sweeper and functional completeness.

I have one more-- well, sort of two more proofs. Only one that I will talk about in detail for Candy Crush and soon enough, Bejeweled. But let's think about Candy Crush is the modern version. I'm guessing you've all played, but in case not, please don't play. It's very addictive.

But on the slides we're allowed to. So you have a grid of colors, colored candies. And a move in this game is to take two candies and switch their order. I guess in this picture I did these two. If you switch these two you get this picture.

And the mechanics of the game, the physics or whatever, is whenever you have three or more candies of the same color in a row or column, or in a row, then they disappear. And anything above them falls. So like this red candy ends up there. In this case, there were no more candies above. In the real game usually there's always more candies above.

But if we're in a small game this won't-- if we're in a big game and looking at a small window, we know everything about what will fall.

AUDIENCE: Do you need to make them fall or do they automatically fall once you get--

ERIK DEMAINE: They automatically fall. Yeah, so they are forced to fall. It's all I get to do is make this move and then stuff happens. So I make this move and then anything that-- any three in a rows or more disappear. And then if that makes more three in a rows they will disappear and so on.

And my hands are off at that point. Once the chain reactions are finished-- not all games work this way, but Candy Crush happens to work this way. Once these things are finished, then I can do another exchange. And when I do an exchange I must make three in a row. I can't just swap these two guys. It's not interesting.

AUDIENCE: What if it's like a T pattern or like non-linear?

ERIK DEMAINE: Yeah, OK. In this picture we'll never have Ts. It will always be three in a row, and never a four in a row. With four in a row, magical striped candies happen and you don't want to know.

So potentially you can get up to five in a row with these rows, but in these reductions that won't happen. So we don't care what the rules say there. We're only going to get three in a rows.

There is a weird catch in this proof. I'm going to mention two proofs from 2014, so all very recent. I guess the game isn't that old. And in this proof we're going to assume if-- in general, there may be multiple three in a rows at once. And the proof is going to assume that they are resolved bottom to top, so you do the bottom-most one first, things fall, then you do the next one, things fall, do the next one, things fall.

That's not how the actual game works. In the game, they all disappear simultaneously and then stuff falls. OK, but ignore that for the moment. The next proof will change the model. OK, this is a much easier one to think about and draw the pictures.

So in that model, here's a variable gadget. The starting point's the same, picture top and bottom. And the idea is either you exchange these two to make three, or you exchange these two to make three.

And the consequence is either the right column falls by three or the middle column falls by three. And that's all you can do local to this gadget. So that's easy. Things are going to get messier.

OK, so you have this picture, variable gadget for XI. Above it I'm going to make one of these two gadgets. Actually, both of them, but one at a time. So in fact, it will actually look kind of like this, stacked on top of the variable which is down here. So variables down here. Either the center column falls or the right column falls.

Now, the idea is this is going to eventually connect to a clause, and this variable-- I should say this is not for circuit SAT, this is from 3SAT-- regular old 3SAT. So that variable appears in a clause either in positive form or negative form. Positive, use this gadget. Negative, use this gadget.

So in the positive case, if this falls by three we get this picture, and then these disappear, and so this guy falls one more. And the idea is that's going to trigger something off to the right. That will be next slide. That will be the wire gadget.

If the center column fell by three, nothing would happen. The purples-- so I should mention how these gadgets are constructed. There's a four color sort of checkerboard pattern in odd rows. It's alternating orange, green, orange, green. In even rows it's alternating red, yellow, red, yellow.

And then in a few places we're going to put the purple ones, because purple's best. And all the action is going to be in the purple candies. These guys are just out there-- and the real game has six colors. We're only using five of them. So cool.

These guys are just sort of filler to make sure nothing else happens. But still things fall vertically in a nice controlled way. OK, so if these fall by three, the purples don't align so nothing happens. In this case, if the center one falls by three we get alignment, and that falls by one.

So great. Also note, these gadgets stack somewhat nicely. In that other than this three gap-- in reality, more candies will fill in here. I drew it this way so you could see how much the columns actually fall.

Remember, this all sort of happens in one move. So this falls, these clear, then this will clear, and stuff happens. It's annoying. So, so far everything's good because each of these has fallen by one in addition to the original three.

But in this case, this follows by an extra one, and I think you need to add an extra candy here. I'm a little unclear in the details on how to do that. It's mentioned in the paper, but without details.

Anyway, in the end we will have a bunch of these stacked on top of each other. Each time the variable occurs in various clauses we're going to have some positive and some negative ones. And we want the same variable trigger-- ah, that's right. There will be-- well, OK. Yeah. I'm going to leave it like that.

So at this point we have a purple thing triggering something to the right. So now I'm going to show you how that part works. This is the wire. So the wire itself is like this. And the idea is that when there's a purple one here, a chain reaction happens. And it happens in this way. As soon as there's a purple candy here, this clears, which makes these two fall by one, which clears that, which makes this fall by one, and triggers the next thing.

This one, some stacking happens. Let's see. So overall, I think this one is actually better behaved. This falls by one here and then two. So overall these columns fell by two.

This column has fallen so far by one, but it's about to fall by a second one, because this will always trigger something. And so these do stack nicely, because you always clear exactly two columns. The worry is that the gadget above it will be skewed.

Let me give you one more gadget, then I'll tell you-- no, actually, I think at this point I should tell you how things fit together, which is this picture.

So we have variables down here in the lower left. Then let's worry about one thing at a time. These are the connection gadgets. I don't think I told you their name. But they are your connector gadgets. This is making a 90 degree turn. From the falling action we're going to get a purple in a particular position, which causes rightward motion.

And then the wires that you saw basically go in some angle off to the right. I want these all to be non-intersecting. And there's sort of two of them. There's going to be the true case and the false case.

And then way over here somewhere, we're going to have a clause where these wires all attach. And then above it we're going to have a reward where you get tons of points, and et cetera.

So let me tell you what happens next. It's a fun way to draw the picture. So above these we'll have some other connections. I should not do it in the same pattern. I should not do it like that. Yeah, sure. Connection, connection.

OK, again, diagonal lines. Great. Not quite diagonal. It's an angle. We have a clause. Everything connects up. Above the clause is a reward. And then I move boards, and that's the picture.

And above the variables, we just keep adding more connections connected to some clauses. And there's not just three variables, of course. There's n variables down here, and we use some, three of them, and then connect that off to a clause.

And they're all just stacked-- I happen to draw the clauses one on top of the other. That shouldn't happen. So there's some extra space here. This is more over to the right. So that none of them will interact with each other. That's the goal.

Except there's some issues, like notice this wire gadget, notice dual rail logic. Either this is active or this is active. And we hope that not both of them are active, although that can actually happen here. I'll get to that.

So when this activates, notice this goes on top of a variable gadget. Now, a wire by itself is kind of good. We've seen it removes two things from every spot. So things don't shift too much. And furthermore, we should hope that either this activates or this activates. So this will shift by a consistent amount in either case.

But the wire may get messed up because the variable actually removes three things from one of the columns. So we need a better wire gadget, which looks like this. This is a super wire gadget. It's really cool. It works no matter how you use it, kind of.

So this is what it looks like in general. And this is regular use. So as before, if we trigger this, then this triggers, and this guy falls by one, just like before.

But now also, if there's a variable down here and this column falls by three-- or sorry. If this column falls by three, so the same picture, the things fell by three, still everything works out. These two fall. These two fall but they meet a different dude.

But there were a bunch of guys to meet up with, so it's fine. And I clearly drew the output in the wrong place. The output should be here I think. Yeah. So in this picture the output was here, also it shifted by one. In this picture the output is here, not up there.

So in both cases it triggers. And finally if the middle column fell by three before this gadget activated, then this, again, triggers this time with this guy being different. Instead of this one matching, this one's matching. That still triggers these guys and this is still the output.

So this is a different wire gadget that works in three different scenarios, depending on what happened below it. So that's good news. That lets us do this type of wire.

And in general, we're going to have a chain reaction all the way down here until we get to the clause from bottom to top. And this we'll resolve. And let me show you the clause. I think details are a little bit messy. There's a little locking mechanism here.

But basically the idea is all of these things come in, x1, x1 bar, x2, x2 bar. All of the half wires or semi-wires. And so we expect one of these to activate. Maybe you don't do that, but that would be weird, and you won't end up satisfying the clause.

So if this one activates, then these align, and you're happy. In general, if you can get this column to fall, you're going to be happy, because above this thing is this reward gadget. And if this column falls by one, you get a million points. And every clause has its own reward of a million points, and million here is like n squared or something.

So you really want to get these. And the only way to do it is to satisfy the clauses. The only way to get all the point is satisfy all the clauses.

Now, the goal in this game, I didn't mention, is to maximize the number of points you can get for a given number of moves. Usually in Candy Crush you have a limited number of moves. So the idea is you're just going to have enough moves to set each of the variables right.

If you end up setting one of the wires directly-- you can trigger any wire by doing a little flip. You can say I'll trigger this one wire. But that will satisfy, at most, one clause. Whereas if you do it on a variable, you're going to satisfy a bunch of clauses, and that's clearly better.

So that's a bit of an argument you need to make. It doesn't help enough. You could even set this wire and this wire, an xy or an x bar wire. But the claim is that will never buy you as much as setting an entire variable, and here's the reason why.

I drew this picture simply to say for every clause we're going to have this picture connected to some variables. In fact, every time you have a clause, you make a million copies of the clause. And million is like n. So that it's really worth setting the var-- every variable appears in millions of clauses. So you really want to set the variables. If you just satisfy one measly clause and get one measly million points, it's not worth it.

But if you can get a million times a million points, whoa, a trillion points. So that's why you should only set variables, not set wires directly.

OK. I think I have roughly covered this proof, except I didn't go up the chain. Here, setting x1 bar doesn't do much except setting one or the other of these ends up shifting all of these things by one, which ends of clearing this, which shifts all of these columns by one.

And basically things look really spread out here, but as you work your way up, they're coming more and more into alignment. The idea is as long as things resolve from bottom to top, you-- it doesn't-- then first you figure out whether x1 satisfies the clause.

If it doesn't, it sets things up for x2. And then maybe x2 satisfies the clause. If it doesn't, it sets things up for x3 here. They're spread out even more. But by the time you've done these things below, that thing will be ready.

So that was the first proof. The second proof-- sorry, before interlude. Candy Crush is based on a variety of games. Most famous is Bejeweled, but there is a long history of-- even Tetris is in there.

There's a long history of match three games. And pretty much all of these are NP-complete. I don't know the details of all of them, but there's this paper by three authors a little bit later in the year that proves basically all of these games are NP-complete, even when you're not given a move limit.

So this is interesting. And Bejeweled, the goal is just to maximize your score. You don't necessarily have a limit on the number of moves, unlike Candy Crush. And also in some levels candy Crush is like that.

So here it was very crucial that we only had enough moves to trigger all the variables. We didn't want to be able to do other things. So here's a different proof, and I'm just going to give the high level picture, basically.

This is cool, because even just figuring out whether you get this one gem, this one candy is hard. In order to do so, you have to line up all of these things for each clause. And here's the reduction from exactly one of 3SAT.

So only if exactly one of the variables get set correctly does this shift to the right level, and only when all of these are shifted to the right level can you trigger this wire to go all the way here and get that guy. And it's the only way to get that guy. You can't do it directly. It can only be triggered from the left-hand side.

And there's also a sequencer over here, which forces all of the variables to get set before this happens. So it's a little tricky, because you don't want it to be, you could say half of them, and then trigger this, and then things happen to align in a cheating way.

The details are complicated, but they actually implemented their reduction. So this is it. Now, as it says here, there are 4,000 rows above and 2,000 rows below. So it's a large construction. And this is for this formula.

[LAUGHTER]

I can clean it up a little bit, remove the-- again, there's a background checkerboard pattern here. So if I generate instance then things are little bit cleaner. And I think highlighted in red are the moves you can make.

So here's the first variable. You could either-- let's do the simple one first-- move this guy with this one, and then trigger, trigger, trigger. All this happens without you able to do anything. And that will move some stuff up top.

Or we can move this one here, and then this is fun. The variable goes down and it just keeps-- it goes down, basically, to where the x1 bar is, down here, and then it triggers that guy, and maybe that guy also. And that one. And--

[LAUGHTER]

Just lots of stuff happens. Anyway, eventually we'll get down to x2 and we'll be able to set it one way or the other.

So the details are obviously a little bit complicated, but clearly they've been explicitly worked out because they can even play the game, which is pretty awesome. Anyway, and they also have the greatest URL, candycrush.isnphard.com. We should try to get other things not as nphard.com.

That is Candy Crush and Bejeweled and all of its friends are NP hard. That's it, unless there are questions.

So we've just started to scratch the surface on the idea that planar graphs are hard, and next class we'll see planar versions of 3SAT and one and 3SAT. If we weren't using here, this is using a non-planar thing. So that helps with a lot of proofs, especially ones that happen in the plane.

So we've seen it with planar circuit SAT. But next we'll do planar and 3SAT, planar one and 3SAT, planar not all equal SAT, those kinds of things. And again, be able to avoid crossovers, which is nice. Cool.