

[SQUEAKING] [RUSTLING] [CLICKING]

**PROFESSOR:** OK, ready for last hour? OK, before I start, was this too slow, too fast? Too slow? Too fast? OK, I have an idea. Everybody, close your eyes.

[LAUGHTER]

OK. Now, you have to say, too slow, too fast, or good. OK, no, you need to close your eyes. Everybody, close your eyes.

OK, too slow? Too fast? Good?

**AUDIENCE:** Good.

**PROFESSOR:** Yes. OK. OK, thank you.

**AUDIENCE:** Hands up. Oh, what is it for?

**PROFESSOR:** All good. It's all good. Passed. OK. OK, great.

OK, so I want to say first, one thing I forgot to mention about the sum-check protocol that we actually use later-- it has a nice property, which is look at what-- so the prover sends low-degree polynomials. What does the verifier send? The verifier just sends random field elements-- so really, randomness. He chooses random  $\log f$  bits and send them over.

This is called a public coin protocol-- public coin, meaning the coins-- so each round, you should think the verifier chooses coins and just sends them publicly to the prover. Here are my  $\log f$  coins. Here they go. This defines a field element, and that's my field element.

This kind of public coin protocols are very useful because we can later, as we'll see in this class, we can use cryptography to eliminate this interaction from this protocol. So the fact that it's public coin, we'll come back to it. But just notice the very first message is truly random. OK.

One more-- Shenian? OK, yeah, sorry.

**AUDIENCE:** I'm just wondering. Are there also protocols with the prover as randomness?

**PROFESSOR:** Good. OK. So as I said, we mentioned the verifier has to be randomized. If it's not randomized, the interactive protocol has no power. The question here was, does the prover need to be randomized? Is it helpful if he's randomized?

And the answer is, I mean, it doesn't help him to be randomized because you can always fix his coins. The verifier doesn't. When is it helpful for the prover to have randomness? When you want to hide. When we do zero knowledge where the goal is to hide thing, then the verifier has random coins that the verifier doesn't know. But in this case, the verifier is not-- we're not trying to protect the verifier.

So why is randomness helpful-- why is it important that the verifier uses randomness? Because the prover needs to send the polynomial. He should not know which element  $t$  the prover, the verifier, will send. If he knew at which point  $t$  will be asked, then he can cheat.

He'll give you a polynomial that happened to coincide with the true polynomial at that point  $t$ . There are  $d$  such points, so he can choose it. He can choose the fake polynomial to coincide.

The power comes from the fact that the prover doesn't know this  $t$ . So we don't trust the prover, and we use randomness against it. The verifier is trusted. We're not trying. So it doesn't help the prover to use randomness in a sense.

Or in other words, if you can fix this randomness once and for all time, think of it as a non-uniform prover. And that's it. The verifier is not going to change his behavior depending on-- because he's honest.

The verifier is always honest. It's the malicious prover that we're worried about. And therefore, the prover doesn't help him to-- he doesn't need randomness because you can always hardwire the best randomness possible to him, and now he's just a non-uniform machine, a circuit, as opposed to a randomized machine. So in this context, randomness to the prover is not needed, but randomness to the verifier is paramount. Yes?

**AUDIENCE:** So when you say public coin, does that mean that you can't see those bits?

**PROFESSOR:** No. Public coin means that the verifier doesn't hide anything. Of course, the prover-- good question.

Thank you. The prover, he never sees the coins in advance. So if he saw the coins before he said anything, he can cheat.

But the verifier never hides coins to himself. So public coin means when the verifier-- the verifier just sends random coins. So I'm the verifier.

You guys gave me a univariate polynomial, a degree- $d$  polynomial. I choose coins, a field element that's random coins. I send it over. I don't keep any state.

All my messages are just random. You give me another polynomial, I'll just send randomness. That's the definition of a public coin protocol. So public coin protocol just means-- so note sum-check is a public coin protocol which means that verifier's messages are truly random-- just random bits.

Or I should say uniform. I just had random, but it's truly uniform bits. That's what public point means. Yes?

**AUDIENCE:** I guess, is it known what happens if the prover does know all the randomness in the  $s$ ?

**PROFESSOR:** Yeah, yeah. If the prover knows all the randomness in advance, he can cheat. Why? Let me tell you how I'm going to cheat.

So I'm a cheating prover. Let's say, before we started, you gave me all your  $t_1$  up to  $t_n$ . Actually, even if you just gave me  $t_1$ , I'm golden. Why?

I faked. I said, the sum is  $B$ ,  $\beta$ . It's not  $\beta$ . It's something else. And now all I want is to find a univariate polynomial that sums to  $\beta$ .

So it's not the true polynomial. It sums to beta, but it agrees with a true polynomial all on  $t_1$  because once it agrees with  $t_1$ , from now on, I need to prove a true statement because I need to prove this, which is actually a true statement. So I'm good. So if I knew  $t_1$ , I can always come up with such a polynomial. The thing is, I don't know what  $t_1$  is.

So actually, I want to give-- Chima actually asked me a very good question in the break. And I want to give nice a geometric interpretation of the soundness proof. So let me give you a geometric interpretation of why this protocol is sound, what's happening in this protocol.

So here's the geometric interpretation. We have a little box,  $h$  to the  $m$ . You can think of it, this is the little box  $h$  to the  $m$ .

I want to prove to you that this low-degree polynomial, if you sum over the value at this point, this point, all these  $h$ -to-the- $m$  points, it's beta. Now let's look at the big  $F$ . I have this big  $F$ .

OK. Now what happens? If you were incorrect, if you cheated me, then this sum-- you sum over all the points. Let's fix this point here, the one possible  $h_1$ .

Maybe what you gave me is false. What you gave me is-- this sum is either false on  $h_1$  or on  $h_2$ . Somewhere, it must be false because it was true on everything, it would be true. So you gave me something that's false on one of them. I don't know which one.

But now I'm going to use error correction to say, well, if it was false on one of them, it's false on a random point,  $t_1$ . So now I forget about this sum. I just look at  $t_1$ .

And I know that the sum only over the rest of the dimension 2 is false on  $t_1$ . And now, again, false on  $t_1$ , that means  $h_2$ . One of them must be false.

I go to a random point in the extension, and there, with high probability, it's false. And this is how I bop up around from dimensions-- so  $t_1$ ,  $t_2$ , to dimension  $t_3$ ,  $t_4$ , and so on, up to the point that I get a random point and left it there. So that's a geometric interpretation of what's going on. OK? OK. Any questions before we-- OK.

So we saw this sum-check. So let's see where we are. We introduced this interactive proof model.

We saw sum-check protocol. Seems very specific. Let's try to see why it's very interesting, the sum-check protocol.

First thing, let me give you an example where the sum-check is useful. And here's the example. Let me show you that #SAT has an interactive piece. IP is the set of all languages that have interactive proofs.

So let me prove to you that #SAT set has an interactive proof. That's an example. This will use the sum-check protocol.

By the way, let me just mention, all the examples that I'm giving you are not just examples. I'm teaching something through it. So don't just, ah, it's an example. I'll forget about it. They're specially crafted.

OK, so what is #SAT? #SAT, we're giving a formula,  $\phi$ . And we're asking, what is the number of satisfying assignments to this formula? That's the question. That's the aim.

So  $\phi$ , you should think of it as a Boolean formula. We're given a Boolean formula. Let's say  $\phi$  is over-- sorry-- over, let's say,  $n$  variables.

And Boolean formula, just think of it's AND and OR. And we have NAND gates. And the question is, how many assignments over 0, 1 to the  $n$  make this formula accept? That's the question.

So by the way, when we say formula, we mean it's a binary. Think of it as a binary tree. So a formula is just a binary tree. Each gate, you can think of it as an AND or an OR. And then the final, final leaves are all  $x_i$  or not  $x_i$ .

So  $\phi$  is of the following form. It's, let's say, AND, then you can have OR or AND, so on and so forth. And at the end, it has  $x_3$ , NAND  $x_7$ , and so on, and so forth. So you can think of the size of this formula as the number of leaves because up to a factor of 2, it's the number of gates, the number of leaves.

It's the same. And just note,  $x_3$  may appear many, many, many times because I really opened it up to a tree. So each variable here can appear many, many times.

So this is what our formula is. And #SAT-- so now I want to say that, let's say,  $\phi$  and  $k$  is in #SAT. It's in the language if and only if  $\phi$  has-- so the number. If you look at all the  $x$ 's in 0, 1 to the  $n$  such that  $\phi$  of  $x$  is 1, this set equals  $k$  if there is exactly  $k$ -satisfying assignments. Questions about this language, free SAT? We're good? OK.

I just want to point out, a formula is different than a circuit in the sense that the formula is a tree. It's a binary tree. A circuit is an arbitrary directed acyclic graph.

And in particular, in a circuit-- in a formula, you need to recompute everything. So if this gate uses a value and this gate also uses the same value, you need to recompute it. You need to compute it twice.

In a circuit, you can do things much more efficiently. So there's also circuits SATs. That's a sharp circuit set. That's a different problem. Here, we're talking about #SAT for formulas, for trees.

So I'm going to give you actually-- so this is known to be a very hard problem. I'm giving you-- think of three sets. Look at the number, like, 10 people a number? How do you know how many satisfying assignments it has? It's considered a very hard problem.

But if you think about it, it's really surprising that you can give an interactive proof for it, which is really nice. And moreover, the interactive proof is really just the sum-check. That's all. That's all it is.

So I'm going to show you how I embed this kind of question into a sum-check. And note that this is just a Boolean formula. Where's the polynomial? I'm going to make a sum-check. But to have sum-check, I need a polynomial. This is just a Boolean formula.

So the idea is to use-- so I'm going to use sum-check for this IP. And the way I'm going to use sum-check is by a technique called arithmetization, which we'll use. So the idea of using sum-check or the idea behind this IP is to arithmetize the formula  $\phi$ .

So I want to prove to you that  $\phi$ , comma,  $k$  is in #SAT. I want to use a sum-check protocol. That's the only protocol I know.

And I want to use it. But I need a polynomial. I need some polynomial in the sum over this polynomial.

What is this polynomial? It's going to be an arithmetization of the set formula. So how do I arithmetize? The idea is very, very simple.

I'm going to convert any-- so this is Booleans all over 0, 1, yeah. I have AND and OR. Everything is over 0, 1.

I'm going to convert an AND gate into MULT gate. So actually, fix any finite field. You can make it very, very big, so you'll have good soundness with respect to finite field  $F$ . So it fits any finite field. It can be very large.

I'm going to convert any AND gate into a multiplication gate. So MULT is just take  $x$  and  $y$  and just output  $x$  times  $y$  in the field. But in particular, what it means-- anything times 0 is 0. And 1 times 1 is 1. It's exactly an AND gate.

I have an OR gate into-- so this is a bit more problem. So actually, let me say-- maybe I'll write it.

So I'm going to write AND of  $x, y$ . I'm just going to write as  $x$  times  $y$ . And I'm going to write OR of  $x$  and  $y$ . Any idea? Yeah?

**AUDIENCE:**  $x, y$  equals  $x$  plus  $y$ .

**PROFESSOR:** Exactly, exactly.  $x$  plus  $y$ -- that's the idea. But of course, then it's 2. So I subtract  $x$  times  $y$ . Exactly.

So of course, 0 goes to 0. If only one of them is 1, you get 1. And 0, nothing.

And if both of them are 1, then this one is 1. So it takes care of it. Good.

And then the NOT-- NOT, I go into NOT of  $x$  is simply what?

**AUDIENCE:** 1 minus  $x$ .

**PROFESSOR:** Exactly. 1 minus  $x$ . So now I have actually an arithmetic circuit.

So this arithmetization allowed me to go from a circuit into an arithmetic circuit or from a formula-- sorry-- into an arithmetic circuit. First, note that each gate is now an arithmetic circuit of size at most 3. It's a tiny circuit.

So each gate really is only blew up by only a constant. And now, really, what I want to do a sum-check over my protocol-- what I want is to look at-- so let's denote the arithmetization by-- so if we denote arithmetized formula by  $\phi$  tilde-- so  $\phi$  tilde goes from  $F$  to the  $n$  to  $F$ . It's just addition and multiplication. It's defined over the big  $F$ . Yes?

**AUDIENCE:** I'm worried that since OR now has  $x$  twice and  $y$  twice, since you said it has to be a tree, do you have to copy and paste  $y$  in order to plug it into that?

**PROFESSOR:** OK. Actually, I don't care. We'll see.

What you're saying is, wait. All I care is that I started with an arithmetic-- with a formula. I don't care in the middle what I do with it.

**AUDIENCE:** So right now, it's not a formula anymore. Is that OK?

**PROFESSOR:** Yeah, oh, sorry. Good. Thank you. Good. Thank you very much. Thank you. Great.

Wow, you guys are good. Thank you. OK, so let's denote the arithmetized version by  $\phi$ . Now, of course, it's natural. What would be sum-check protocol that makes sense to do at this point? Any ideas? Hmm?

**AUDIENCE:** Replace  $F$  with  $\phi$ .

**PROFESSOR:** Exactly, exactly. Let's prove that  $\sum_{i=1}^n \phi_i$  is equal to  $k$ . Why that makes sense? The only thing we need to be careful with is, what is the degree of this thing? So now let's just argue that the degree is not too big.

But if I convince you that the degree is small, then we're done. We just do a sum-check. And this is exactly what you want to prove. So let's just make sure that the degree-- so  $n$  here, like the number of rounds is going to be  $n$ .

There's going to be a degree  $d$ . We just need to ensure that  $d$  times  $n$  is much, much, much smaller than  $F$ , which governs-- I put here the-- which govern the soundness. So let's just see the degree.

So here's a claim. The degree of  $\phi$  is at most-- or let me write it differently. The sum-- let's look at the degree for each variable. The sum of the degree for variable  $i$  of  $\phi$ -- so we have  $n$  variables,  $x_1$  up to  $x_n$ . Each one has some degree.

The sum of these degrees, I'm going to argue, is, at most,  $s$ ,  $s$  being the number of--  $s$  size of  $\phi$ , which is defined to be-- this is by definition-- the number of leaves in  $\phi$ . So let me argue that the size is, at most,  $\phi$ -- the degree is, at most, the sum of the degree is, at most,  $s$  or the size.

By the way, before actually I do that, just to make sure because I actually noticed that I glossed over too quickly-- I said that this is all we need to prove. But I just want to make sure. Is it clear that  $\phi$  and  $\phi$  on  $\{0, 1\}^n$  is actually equal to  $\phi$ ? Yeah, that's very important because we want to make sure that we're actually summing over the actual  $\phi$ , like in  $\{0, 1\}$ , because we want to prove #SAT of the  $\phi$  formula.

And the reason is, as we said, is that on  $\{0, 1\}$ , we really behave like the AND and OR. So of course, outside of  $\{0, 1\}$ , who knows? We're over a big field. Who knows what's going on? But on  $\{0, 1\}$ , we behave exactly like the AND and OR, like the original tree.

And therefore, this arithmetic formula is actually equal to the original formula on  $\{0, 1\}^n$ . Yeah? OK. So good.

So this is what we want to prove. Now let's just look at the degree. Yeah?

**AUDIENCE:** What degree  $\phi_i$  is this?

**PROFESSOR:** Good, good, good. This just means this is the degree of  $x_i$  in  $\phi$ . Or in other words, I can argue that each variable has degree, at most,  $s$ . But I can argue something stronger. The sum of the degrees of all the variables is, at most,  $s$ .

Note that  $s$  is polynomial here. That's the input. The input is a set formula. It's polynomial.

So as long as this is the degree, we're happy. The communication complexity, the verifier complexity is going to be  $n$  the number of variables,  $d$ , which is like  $s$  polynomial, and  $\log$  in the field. So the verifier is going to be polynomial.

Now we care about-- now there's a notion of polynomial time verifier. But if the degree is, at most,  $s$ ,  $s$  is polynomial. We're good. OK.

So why is this the degree? So to prove that this is the degree, the proof of this claim is by induction. It's actually a pretty straightforward proof. And it really uses the fact that  $\phi$  is a formula. The original  $\phi$  is a formula. So let's see.

So the proof is just by induction. And here's how-- OK. So we're going to prove by induction on the depth of this formula. If we have just one gate, then, OK, it's clear the degree is just 1. So that's fine.

So if number of gates or the depth if you want is 1, then, of course, the degree is going to be less than the size. It's degree is just 1. That's if you just have a single gate.

And now if you have a more general-- so now suppose you have a circuit of the form, let's say, ADD or MULT-- it doesn't matter-- or AND, you can have  $\phi_1$ . Suppose  $\phi$  is  $\phi_1$  AND or OR-- I don't care--  $\phi_2$ . This can [INAUDIBLE] at the AND. But of course, it can be also an OR.

Then how do we arithmetize this? So now what is  $\phi$  tilde?  $\phi$  tilde is going to be either  $\phi_1$  tilde times  $\phi_2$  tilde. Or in the case of-- this is for AND. If it was OR, then it would be  $\phi_1$  tilde plus  $\phi_2$  tilde minus  $\phi_1$  tilde times  $\phi_2$  tilde-- all, of course, over  $x_1$  to  $x_n$  over  $x_1$  to  $x_n$ . Yeah? OK.

So let's look at the-- I want to argue that the arithmetic version is of the sum of the degrees of all the variables, at most,  $s$ . So let's see. Let's look at  $\phi$  tilde.

If the original  $\phi$  was AND of  $\phi_1$  and  $\phi_2$ , then  $\phi$  tilde is just the multiplication-- by definition, we replaced  $n$  with multiplication-- of the  $\phi_1$  tilde polynomial times  $\phi_2$  to the polynomial.

But what do we know here? Here, the degree of each-- so now let's suppose-- OK, suppose  $\phi_1$ -- let's say it's of size  $s_1$ . And suppose  $\phi_2$  is of size  $s_2$ . Now we know that the size of  $\phi$  is equal to  $s_1$  plus  $s_2$  because it's a tree.

So there's no kind of-- everything is open. So the left sides of the tree is  $s_1$ , size  $s_1$ . The right side of the tree is size  $s_2$ . The entire tree is the size 2, which is  $s_1$  plus  $s_2$ .

So what can we say in this case? So let's focus on the AND. The OR is exactly the same. So what can we say about the sum of the degrees? So sum of degree-- the degree of the  $i$ -th element-- of  $x_i$  in  $\phi$  is equal sum and  $i$ . What is the degree of the  $i$ -th variable in  $\phi$ ? It's the degree here plus the degree here.

When you multiply things, I need to sum the degrees. So it's the degree of  $x_i$  in  $\phi_1$  tilde plus the degree of variable  $i$  in  $\phi_2$  tilde. We have two polynomials. Multiply the degree  $x$ . But this is just  $s_1$  plus  $s_2$  by induction-- by the induction hypothesis.

OK, so the degree here-- the sum of the degrees is like the size of the formula. We actually don't really care about the sum. We care about the degree itself. But this shows you at least the degree of each one is also, at most, the size of the formula. And therefore, the sum-- so now when we run-- so let me put this back up.

So we run the sum-check on this polynomial, and we get the communication complexity. And the verify runtime will be  $n$ -- or the number of variables here,  $n$  times the degree, which is  $s$ -- essentially the size of the input, that's  $s$ -- and  $\text{polylog}$  the field. So you just need to take-- and similarly-- and the verify-- yeah, the degree, which is  $s$ , the completeness is always 1.

And the soundness-- importantly, the soundness of this protocol is  $n$ , the number of variables, the degree, which is  $s$ , divided by the size of  $F$ . So all we need to make sure is to arithmetize with a field that's significantly larger than  $n$  times  $s$ . And this will be the soundness here.

And of course, then you can repeat all this to get here as long as you want. OK. Any, yeah, questions? Yeah? You can go first.

**AUDIENCE:** OK. So when you do the sum-check, when the sum is happening, is that necessarily a sum over the field? Or can it be any sum over a different field? If your original field is just, I don't know, some prime field.

**PROFESSOR:** Just note, in this specific problem, I don't have an original field actually. For this specific #SAT, I'm over 0 0, 1. There's no field. It's not an algebraic.

It's a Boolean thing. I don't have any polynomial, any algebra, any field, nothing. I just have 0, 1, and I have AND and OR gates.

And to prove that the number of satisfying assignment in the Boolean cube is  $k$ , I choose a field of my liking, and I arithmetize. I think of this circuit as polynomials, as having addition and multiplication gates. And in my head, I think of this circuit as this formula as a polynomial.

So I choose the field. It's not actually the problem. I, on purpose, chose a field so that I can do the sum-check protocol.

**AUDIENCE:** Right. Yeah. But I guess what I'm asking is, for the sum in the sum-check, let's say not for this problem.

**PROFESSOR:** Oh, I see. I see. You're saying, in general. You're saying, look, let's say you have a sum-check problem. Someone gave you sum-check problem. And you're not happy with the field.

The field is not good enough for you, or it doesn't give you enough soundness. Can you amplify? Can you work over a different field? Is that what you're asking?

**AUDIENCE:** Or I guess what I'm asking is, if you have sum-check and there is some field-- let's say you're working mod, like, some large prime or something. But then, will the sum that you're doing be also over that field, necessarily? Can you just change those arbitrarily?

**PROFESSOR:** Good. So when I do a sum-check over the field, everything will be over that field, yes. Everything will be over the field. However, that said, if I'm not happy, the  $p$  of-- you're saying, so I got a sum-check protocol with certain  $p$  over  $\text{gf}_2$  or  $\text{gfp}$ . So everything is mod  $p$  with certain  $p$ .

Now, yes, everything should be over this field. Now you can say, well, this  $p$  is not that great because it's not that much bigger than  $m$  times  $d$ . And I'm not really happy with this field. Or maybe even it's smaller than  $m$  times  $d$ , for all I know. It's not a good field, not big enough. Then what do I do?



So I'm saying, it's OK. Don't worry. Just take an extension field. Instead of looking over  $p$ , you work over  $\text{gfp}$  to the  $n$ .

So you can take a field of size  $p$  to the  $n$  for any  $n$  that you want. And you're only going to pay polynomially or linearly with this  $n$ . And then you can amplify your soundness.

**AUDIENCE:** OK. I was just wondering because I was wondering what happens if  $k$  is larger than the size of the field or something?

**PROFESSOR:** Oh, I'm going to choose the field to be much bigger.

**AUDIENCE:** OK, I see.

**PROFESSOR:** Yeah, I'm going to choose the field. Otherwise, you can't-- yeah. I need to choose the field to be much bigger than  $k$ . Yeah?

**AUDIENCE:** So like--

**PROFESSOR:** Oh, sorry. I forgot you. Yeah. OK, yeah?

**AUDIENCE:** This is related to this point, but we need to choose a field that has really big characteristics, and we to choose a really big prime number. We can't just raise it to the  $n$  because  $k$  can be  $2$  to the  $n$ .

**AUDIENCE:** Can you find field that's bigger than  $k$ ?

**PROFESSOR:** Yeah, no, but the question is, can you-- no, but the question-- well, OK. In our case, you can just choose the field to be as big as you want. In the case where you're giving a sum-check, of course, the sum is in the field.

It's something about the field. You're giving something about the fields. So, OK, we'll always be in the field.

**AUDIENCE:** But for this #SAT one--

**PROFESSOR:** For this specific one, I'm thinking of, yeah, you choose a  $k$  that's bigger. However, I'm not 100% sure that-- hey, let me think. Can you use small characteristic?

**AUDIENCE:** Can you Chinese--

**PROFESSOR:** Yeah, that's what I'm thinking. I'm not sure if you can't use some Chinese remainder trick and use over a small characteristic. I need to think about it.

But anyway, in this case, just choose a  $f$  to the  $\text{gfp}$  for a large  $p$ , and that's bigger than  $k$ . Yeah. Sorry. Did you have a question?

**AUDIENCE:** I was going to ask, is this the only way we know how to do for the #SAT that's easy?

**PROFESSOR:** This is easy. What do you mean? You want easier?

**AUDIENCE:** It's easy. It's nice. I'm just wondering.

**PROFESSOR:** So no, I think this is the simplest way. So let me just say-- so we put #SAT into IP in this way. You can ask, what about, let's say, sharp circuit SAT? Which is the same thing, but we're given a circuit.

So we're given a circuit, and you want to know how many satisfying assignment does it have over 0, 1 to the n. It's the same thing, except for phi is now-- it's not a tree anymore. It's a directed acyclic graph.

And it turns out that it also has an IP because it's in p space. You can go over all possible x's--  $x_1$  up to  $x_n$ -- and check if it's satisfying or not. And it's small space. So it's in p space, and so you can ask, OK, so what about this? Now you can do sum-check because the degree won't be small.

So actually, you can do it. But that's much. Much more complicated and requires this degree reduction thing. That's how Shamir proves IP equals p space. It's that there's some degree reduction there that really complicates matters.

So you need that for a circuit set. But for SAT, because it's so low degree immediately, it's nice. And that's all you do. Yeah, you should be-- where's the gratitude?

[LAUGHTER]

**AUDIENCE:** So I guess, the one interpretation of this question, I suppose, is that suppose you're given another problem. It's a permanent or something. So I guess, it's a formula you define. But I want to direct without reducing to a #SAT and so forth. I mean, I want a direct protocol. It's not very well-defined.

**PROFESSOR:** Yeah. So I have to say, almost all the protocols I know for IP go through sum-check.

**AUDIENCE:** Actually, there's this weird one by-- I might be wrong. I hoped that Irena Mano was one of the authors, not the other one. But this is a weird one that goes through a quantum problem. If they were trying to prove verification, information theoretic verification or BPP and they failed, but they got sharp peers in IP in some other way.

**PROFESSOR:** So I'm not familiar with that. But I think the main ones go through sum-check.

**AUDIENCE:** [INAUDIBLE] sum-check?

**AUDIENCE:** What I remember is there's this-- it's a little like sum-check. But they're also looking at reduced densities over updates and stuff.

**PROFESSOR:** Any other questions about #SAT? Yeah?

**AUDIENCE:** Sum-check protocol is-- I believe, it's operating over a field, like all the summation goes to the field.

**PROFESSOR:** Right, right, right. Yeah, yeah, yeah.

**AUDIENCE:** So does that mean that we'd be safe to make sure to ensure correctness, we must ensure that the field has characteristic greater than the number of satisfying solutions?

**PROFESSOR:** Yeah, exactly.

**AUDIENCE:** In this case 2 to the n.

**PROFESSOR:** Yes. Yes, but the verifier runs in time log in this circuit. So in the log in the-- sorry, in the size of the field.

Yeah. Log in the size of the field. So the field can be 2 to the n.

**AUDIENCE:** Yeah. So that's a requirement. You can't fix small characteristic.

**PROFESSOR:** Yeah. So yeah, OK. So what we should do is take gfp, large field, like, large characteristic. I'm not sure you can take small characteristic, actually. I need to think about it because maybe some Chinese remainder theorem can help you.

**AUDIENCE:** But that was quite expensive not using the p. What's [INAUDIBLE] protocol over many small primes and make sure that the answers are all the number of suspects.

**AUDIENCE:** Oh, I see.

[INTERPOSING VOICES]

**PROFESSOR:** Yeah, exactly.

**AUDIENCE:** Multiple times.

**PROFESSOR:** Yeah, exactly, exactly.

**AUDIENCE:** You need to target those primes to be larger than k anyway.

**AUDIENCE:** Yeah, it does. Yeah.

**PROFESSOR:** Why doesn't it help?

**AUDIENCE:** Because--

[INTERPOSING VOICES]

**PROFESSOR:** Yeah, but--

[INTERPOSING VOICES]

**PROFESSOR:** Yeah, yeah, yeah, yeah. No, but if you don't want to, sometimes--

**AUDIENCE:** Actually, I'm not so sure because the computational complexity is going to be-- is probably going to be smaller. Instead of log, a big field size to the power of 3 or something, you have sum of log small-size fields to the power of 3, which is-- I see. Yeah.

**AUDIENCE:** And you can parallelize.

**AUDIENCE:** You can probably see.

**PROFESSOR:** Yeah. OK, so by the way, just note about the proof of runtime here, of course, it's large because it needs to find the-- I mean, what can you do? You need to at least count the number of satisfying assignments. It takes  $2^n$  to the n, at least, to find them. And then-- OK.

So far, we gave some examples. So where are we? We talked about IP. We showed the magical, amazing sum-check protocol. We showed how to use the sum-check protocol to give an IP for an inductive proof for #SAT. In

In some sense, maybe you're not so blown away at this point because, actually, #SAT, like a formula, is really going to-- arithmetization is not like, wow, we know we all know that  $n$  is like multiplication, and AND and OR can be written as small arithmetic thing. So it's actually an arithmetic circuit. So it actually gives you a set of low-degree polynomials.

So it calls for something. It is a sum-check. Without much change, it presents itself almost as a sum-check.

So maybe we're not convinced of the generality of the sum-check. And the next thing I want to do is show you actually a completely different thing that doesn't seem to be at all low degree or anything, and try to show you how you can use the sum-check. And actually, this takes us at a nice way into the idea of doubly efficient proofs.

So far, when we talked about interactive proofs, our focus was the verifier is efficient. He's polynomial time. The prover can be whatever.

And indeed, in the protocols, both sum-check and #SAT, of course, the prover runs in exponential time. He has no choice. He needs to do the computation. OK.

The idea that-- so now let's shift gears and talk about the notion of a doubly efficient interactive proof. So the notion of a doubly efficient interactive proof-- the idea is actually, you know what? So the main idea is we care about the runtime of the proof.

So the prover power is not all-powerful. It brings us to real life. It's very nice that the verifier is polynomial. The prover is all-powerful.

But guess what, all-powerful doesn't exist. Everything is limited, to some extent. There's more powerful and less powerful, but there's not all-powerful.

So what we want is, yes, the prover is more powerful than the verifier. Otherwise, we don't need it. The verifier doesn't need him. But even the prover is not all-powerful.

So in a doubly efficient interactive proof, we want to say, the prover has a problem at hand. And he should not spend too much overhead in convincing the verifier that his solution-- let's say that  $x$  is in the language. And this is interesting even if the problem-- so when we talk about doubly efficient interactive proofs, we even care of problems in  $P$ .

So often, when we talk about interactive proof, we think, even, let's say, the problem is polynomial time. So I want to argue proofs of polynomial time, a language. But I want to verify. Let's say the time to determine this language is, I don't know,  $n$  to the fourth. I want the verifier to be linear time.

So the idea is there should always be a gap between how powerful the prover is, and the verifier is weaker. But I really care about both. The verifier should be very, very efficient. And the prover should also be somewhat efficient. That's the idea.

OK, so what is a doubly efficient interactive proof? So a doubly efficient-- let's say, for example,  $L$  in some  $D$  time  $T$ . So it can be computed in deterministic time  $T$ . Again, you can think of  $T$  even to be  $n$  to the fourth.

It can be polynomial. Is an IP where the prover runtime-- the honest prover-- the honest prover's runtime is only polynomial overhead in the time of determining the language. So it takes time  $T$  to determine runtime, at most,  $\text{poly } T$ .

And the verifier runtime should be much, much less. Ideally, we want the verifier to run in time almost linear or quasi-linear-- so  $n \text{ polylog } n$ , let's say, which is denoted by  $\tilde{O}(n)$ . So ideally, we want the verifier to run in linear time or quasi-linear time. And the prover should not have too much overhead.

Actually, ideally, the prover should also have just linear overhead. But definitely, by definition, it should not have more than polynomial overhead.

**AUDIENCE:** So again, here is just [INAUDIBLE].

**PROFESSOR:** And sorry, yes, yes. Thank you. This is  $T$  of  $n$ .  $N$  is the instance size. Yes.

So of course, the verifier needs to read the instance, but not do much more than that. So it's just an interactive proof where we care about both the prover and the verifier. OK. And yeah, and so let me mention that these things that people actually care about because they use it in these blockchain protocols and so on.

And so they really actually-- a lot of work that's done on this-- and I think in the lecture notes of Justin Taylor, I'm now going to refer to these sections. We won't cover them, but they exist sections where he really talks about the exact parameters, the exact numbers, the exact implementations, and so on, and so forth because these are things that actually people care about because they want to run it. So actually, we really care about the overheads here in practice, at least.

So far, we said the sum-check is really-- of course, it seems like outside the realm, even, of double efficiency. But actually, now let me show you a specific example, which, again, is beyond just an example because we'll use it to teach something where it's actually in  $P$ . And I'll show you an interactive proof for this problem using the sum-check protocol-- a doubly efficient interactive proof for this problem. Yeah?

**AUDIENCE:** They're different roots, like  $T$  and the other  $D$  time.

**PROFESSOR:** Good. Yeah, yeah, yeah. So take any problem, any language, for which membership in this language can be decided in time  $T$ .

So these empty means there is a deterministic algorithm that runs in time  $T$ . What  $T$  is-- it can be anything. You can think of  $T$  of  $n$  to the fourth,  $n$  to the 10th.

I don't know.  $N$  to the  $\log n$ , quasi, poly, whatever. All I care-- so of course, the prover needs to compute. If he's proving something, he needs to know that it's true. So he needs to do this deterministic computation.

So of course, he needs to run in time  $T$ , but should not run in much more than  $T$ . And by not more, theorists say,  $\text{poly } T$ . Applied people say linear in  $T$ . The real applied say not more than 128 or whatever. But yeah, depending how far you are in that spectrum.

OK, so good. So now let me show you how a doubly efficient interactive proof for counting the number of triangles in a graph. That's the example.

Next week, I'm going to show you actually interactive proof for all-- doubly efficient proof for all low-depth circuits. So I'm going to generalize it by a lot. But let's start with an example.

So let's say we're given a graph  $G$ , and we want to count the number of triangles in the graph. So given  $G$ -- so we want an interactive proof for counting the number of triangles in a graph  $G$  with vertices  $V$  and edges  $E$ . OK.

So I want to cast it as a doubly efficient subset protocol-- kind of subset protocol, but-- so notice this is a problem in  $P$ . I can go over all possible three vertices, triple to vertices, and check if there's a triangle. So I can check this from time  $n$  to the third.

I want that my verifier to be linear. So I want a prover that runs in time not much more than  $n$  to the third. Or let's say polynomial time to convince a linear verifier that this graph has exactly  $k$  triangles. And I want to cast it somehow in a sum-check protocol. So how do I do that?

So first, I need to come up with a polynomial. I need to embed a polynomial here. Not, there's nothing polynomial-like here. Talking about graphs.

So remember how I told you everything is a polynomial? Everything can become a polynomial. This is an example that shows you why you take anything and make it a polynomial.

So what is the idea? Let's look at the adjacency matrix corresponding to this graph that checks if there's an edge between a pair of vertices. Now, we can write this matrix as a function  $f$  that goes from, let's say,  $V$ , like number of vertices times number of vertices to  $\{0, 1\}$ . It takes a pair of vertices, output 0 if there's no edge, 1 if there's an edge.

Let's denote the number of vertices here by  $n$ . So sorry, I denoted there the input length by  $n$ -- close enough. It's not really the input length, the number of vertices, but it's denoted by  $n$ .

So now we can think of the adjacency matrix as a function. So think of the adjacency matrix-- let me write it as a function,  $f$ , that goes from  $\{0, 1\}$  to the log  $n$ . I'm going to assume that  $n$  is a power of 2.

So I'm going to think of a vertex. So  $v$ , as I said, is of size  $n$ . I'll write it here, so it won't be in my way.

I'm going to think of the function as taking a pair of vertices where I represent a vertex as a vector in  $\{0, 1\}$  to the log  $n$  and any number of  $n$ , like binary representation of the-- I think of the vertices as numbers between 1 and  $n$ . I'm thinking of the binary representation. And I output 0, 1 whether there's an edge or whether there's no edge between these two vertices. So this says 1 if and only if there exists an edge between the two vertices.

Now, suppose you want to argue that the number of triangles is, let's say,  $k$ . So what do you need to prove? You need to prove that the sum over all  $i, j$ , and  $k$  in  $n$ -- or let me even make myself in  $\{0, 1\}$  to the log  $n$  to make it easy for myself because I'm representing in binary.

So for any three vertices, let's count the number of triangles. So when there's a triangle, if  $i$  and  $j$  are connected, this is one, and  $f(j, k)$  is connected, this is one, and  $f(i, k)$  is connected, this is one. They all need to be connected in order for it to be a triangle.

So if it's a triangle, I get 1. If it's not a triangle, I get 0 because one of them is going to be a 0. So I'm going to count the number of triangles. And I want to prove check that this is  $k$ -- almost, almost. I lied a little bit.

I overcounted because there's permutation--  $i, j, k$ ;  $j, k, i$ . So I need to divide by  $1/6$ , which is the number of ways to permute any  $i, j, k$  because any  $i, j, k$  is counted like all permutations. So this is what I want to prove. I want to prove that the sum of all possible triangles  $i, j, k$ -- I counted each  $i, j, k$  here essentially six times, so I need to divide by  $1/6$ .

The sum, the number of triangles, is  $k$ , almost. At least, I have some. It's getting to look like a sum-check, but there's no algebra. I just wrote in numbers.

I mean, I took binary representation. But at least, I have some sums. So if I had some nice algebra here, if this was all over a finite field, and let's say this  $f$  was also low degree, then I can use sum-check.

So again, suppose this was over a finite field. And suppose this  $f$  was very low degree, and the field was large, so we have good soundness. I want to prove sum of this big  $F$  is  $k$  sum-check. So I just need to tell you how do I get to-- how do I add algebra to make this degree a low degree? And where does the finite field come from, and so on?

I'll do that. But before I'll do that, also note the sum here is only over  $0, 1$  to the  $3 \log n$ . So remember, the problem was the prover runtime is very large.

It's  $H$  to the  $m$  times the time it takes to compute one function.  $m$  times this  $n$ . This  $m$  is going to be small. But the  $H$  to the  $m$  is the problem.

But here,  $H$  to the  $m$  is just  $2$  to the  $3 \log n$  to the  $\log n$ . So it's poly. So if I'm able to put a sum-check like algebra here, I will get double-- the proof will still be polynomial time, which is what I want. The question is, how do I put algebra here?

So this is where-- so I'll say a little bit about this. And then we'll break for today and continue next week. But there's a generic way to convert any function into a polynomial. And the way to do it is via a technique called low-degree extension.

So if you actually remember, I started class with an example of the matrix multiplication. And if you remember the matrix multiplication, also, there's no polynomial. It's a matrix, nothing.

But what we did-- the randomized algorithm that we showed, we thought of a row in the matrix as a polynomial. So we took a row, which consists of  $n$  elements, and we thought about it as a univariate polynomial of degree  $n$ . Remember, we thought of the row as coefficients of the polynomial.

So we thought it was a polynomial of degree  $n - 1$ , I guess. There were  $n$  elements. One is the free coefficient, so degree  $n - 1$ .

So we started already the class with an example that I have a row-- nothing, no algebra, nothing, an arbitrary row of matrix. And we converted it to a univariate polynomial of degree  $n$ , like the length of the row. What we do here is a very similar idea, but in the multivariate setting.

So what we can do-- we could have taken any  $n$  elements-- that row, for example. Or in this case, the elements are all the kind of-- for any  $F$ , you can think of it like a matrix. It's like  $n$  squared elements, if you want, for any two elements, whether there's an edge or not an edge. So you can think of it as  $n$  squared elements. It doesn't matter.

We can take any set of elements and convert it either to a univariate polynomial, as we did before of that degree. Or that's not good for us because here's the degree is going to be  $n$  squared. The verifier runs and the degree and more. It's not going to be efficient enough.

But the idea is we can convert it to a multivariate polynomial with much, much smaller degree. And that's what's called the low-degree extension. So whereas before, we just did a univariate polynomial-- so we had to put the  $n$  elements as coefficients. Each one added a degree, if you will.

If we put it in a multivariate polynomial-- so if we think of it as a polynomial over, let's say,  $H$  to the  $m$ , then we can stack inside  $H$  to the  $m$ -- we can stack  $n$  points. We can make  $H$  and  $m$  much smaller-- each  $H$  and both  $H$  and  $m$  to be much, much smaller than that. And that's what we're going to do. So let me explain this more and more precisely. OK.

OK, so here's the theorem we're going to use for this. For any function, not polynomial, can be arbitrary, like the adjacency matrix function-- for any function  $f$ , there it's from  $0, 1$  to the  $2 \log n$ . But let me take it more generally from  $H$  to the  $m$  to  $0, 1$ . Again, in this example,  $H$  is just  $0, 1$ . And  $m$  is  $2 \log n$  because we have there  $2 \log n$ .

But let me give you something more generally. So take any function. This is essentially any set. You can think of this as any set of this-- or this is equivalent to any element in  $0, 1$  to the  $H$  to the  $m$ .

So it's just a binary string of length  $H$  to the  $m$ . So take any binary string of length  $H$  to the  $m$ . It just tells you, for each element  $H$  of  $m$  what the value is,  $0$  or  $1$ .

So for any function and any finite field  $F$  that contains  $H$ -- of course, in that example  $0, 1$ , any  $F$  contains  $0$  and  $1$ , so that's not a problem. There exists a unique-- what's called low-degree extension. I'll explain what this means in a second. It's a function  $f$  tilde from  $F$  to the  $n$  to  $F$  such that-- so in what sense, is a low-degree extension. First, it's an extension, namely  $f$  tilde.

If you look at an  $H$  to the  $m$ , for the element and the little cube, it's exactly  $f$ . So it extends. It extends  $f$ .  $F$  was an  $H$  to the  $m$ .

$F$  tilde is in a bigger cube. It extends. So on the small cube, they agree.

The second thing is that it's low degree. So it's a low-degree extension-- an extension and a low degree. How low?

So  $f$  tilde is of degree  $H$  minus  $1$  in each variable. So again, what is the theorem? The theorem is you can take any set-- if you want, you can take any set. So let's say I have  $n$  numbers. Take any fixed-- so take any set.

Let's say fix any vector. Let's say  $w$ , and let's say  $0, 1$  to the  $n$ . You can always embed it. You can think of  $w$ -- and an equivalent way is to think of  $w$ -- oh, sorry.



I meant to say, sorry, as a field. Let's say  $w_1$  up to  $w_n$  and  $0, 1$  to the  $n$ . You can embed it. You can think of it as a function from  $H$  to the  $m$  to  $0, 1$  as long as  $H$  to the  $m$  is  $n$ .

So you have a long sequence of  $0, 1$ 's folded into a nice cube of size  $H$  to the  $m$ . So essentially, you should think of the  $H$  representation of  $n$ -- the  $H$  representation. So  $H$  to the  $m$  is equivalent to the set. You can think of a bijection between that and all the numbers between  $0$  and  $1$  to between  $1$  up to  $n$ .

And now the claim is, actually, you can take this-- so this is the, let's say,  $H$  by  $m$ . You can extend it to a bigger function,  $\tilde{f}$  over  $F$ . This is over  $F$  to the  $m$  such that it has degree  $H$  minus  $1$  in each variable.

So just note, if we think of  $m$  as being  $1$ , the univariate case is exactly what we had before. If you had just a string of size-- if  $n$  is  $1H$ , essentially is like  $n$ . I take  $n$ .

I just expand it in this way to a degree  $n$ . But then the degree is very large. That's not good. The verifier runs in time the degree. That's awful.

So I make the degree much smaller. How do I make the degree much smaller? I add more variables. So the idea is add many more variables. Now in each dimension, the degree is only  $h$  because now I don't need the size of  $H$  is  $n$ , but  $H$  to the  $m$  is  $n$ .

And now in each dimension, the number of elements is like  $H$ . And the unit on each dimension, the degree will be  $H$  minus  $1$  instead of  $n$  minus  $1$ , like it was in the univariate case. I'll prove all this, but this is just a high-level idea.

So again, the high-level idea-- we'll go into the proof next time of this theorem. But the high-level idea is we all know by the univariate case that you can take any string of length  $n$ , encode it as a univariate function of degree  $n$  minus  $1$  by-- there's two ways of doing it-- either by thinking of the string as the coefficients. That's one element, one way to do it.

Another way to do it, which is how the low-degree extension works, is by thinking of the string as the value of the polynomial in, let's say, polynomial on  $1, 2, 3$ , up to  $n$ . Let's say  $1$  up to  $n$  is in the field. So think of gfp, let's say.

Here, what we do-- we take your points. We take a polynomial  $F$  for which your points are sitting in this small cube. So let's say you have a vector of size  $1$  up to  $n$ . We'll put here  $w_1, w_2, w_3, w_H$ -- all the  $w$ 's we put here.

And then the theorem says there's a way to extend this entire thing into a bigger polynomial of degree  $H$  minus  $1$  in each variable such that the-- if you look at the polynomial in this cube, it's exactly what we had before. So you have your vector or your function from  $H$  to the  $m$  to  $0, 1$ . Here and the rest, it's just extended and extended in a way that the degree of each variable is  $H$  minus  $1$ .

So for example, if you want  $H$  to the  $m$  to be  $n$ , what you can take, for example, is  $H$  to be  $\log n$ . The degree is only  $\log n$ , which is great. And you can take  $n$  to be even smaller than  $\log n$ . If you do the math, it's going to be  $\log n$  over  $\log \log n$ .

And note that the verifier runs in time that depends on  $m, m, H$ , and  $D$ . And  $D$  is going to be also  $H$ . So all of this is going to be poly log, which is great.

So next time, what we're going to do is prove this low-degree extension theorem and then show how to use it first for the triangle. That would be pretty immediate. And then we'll show how to use it for a general low-degree, low-depth circuit. OK, thanks.