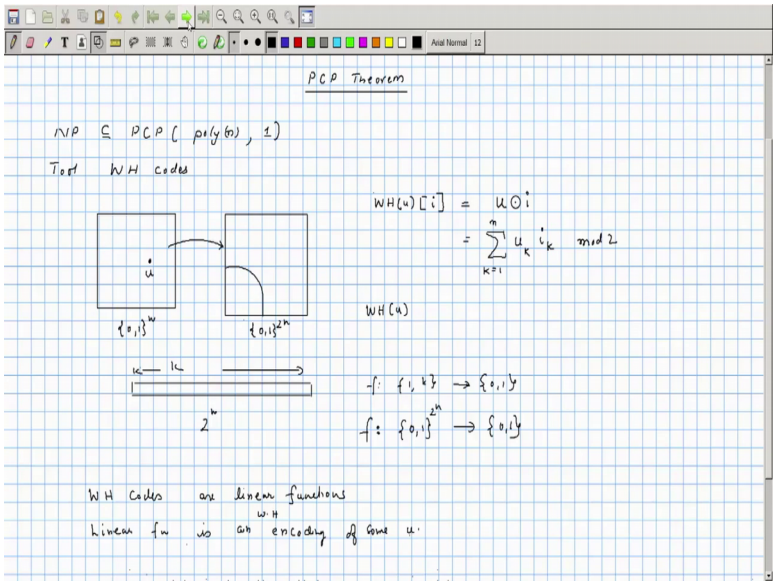
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 37
Probabilistically checkable proofs - II

(Refer Slide Time: 00:29)



So, we are learning about PCP theorem. We will prove the weak portion, the weak portion of PCP theorem states that NP that is contained inside, PCP with the amount of randomness that we use is poly n and the number of queries made is constant and the tool that we will be requiring as what is known as Walsh Hadamard codes. So, Walsh Hadamard codes are nothing but I think of them it is a. So, let us look at strings of length n.

Each of them is mapped to a string of length 2 to the power n in a certain way. So, this is 0, 1 to the n this is 0, 1 to the 2 to the n, ok. So, there is a mapping. So, only very few words amongst this collection are strings of length 2 to the power n will be Walsh Hadamard codes. The mapping was as follows, WH corresponding to a particular string of length n let us call it as u, WH of u will essentially be a string of length 2 to the n and at the ith position of the string which will denote by WH u of i. This is going to be equal to u product i and this product is nothing but summation over k varying from 1 to the length u k i k. So, ith position the sum number between 0 and 2 to the power n minus 2 to

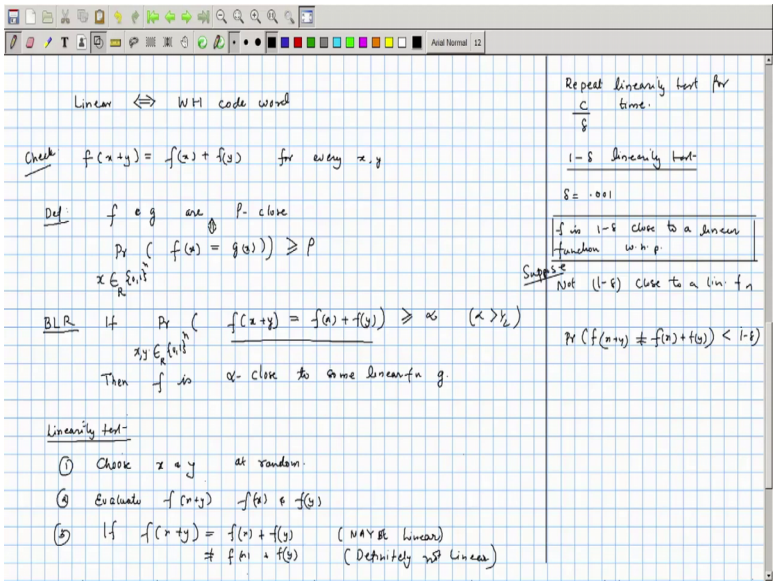
the power n minus 1 and that is a big string of length n the kth bit of that is i k, ok. So, if you take this dot product mod 2, that is the value at the ith position, ok.

So, any string of length n is converted into a unique code word of length 2 to the power n and that code word is called as the Walsh Hadamard encoding of u, ok. So, by WH u we mean the Walsh Hadamard encoding of the string u. You can also view these strings in 0, 1 to the 2 to the n as linear functions, ok. So, look at any string of length 2 to the power n, ok, any binary string you can essentially view it as a function from see these positions are they were k positions you can think of it as a map from 1 to k to 0, 1, ok. So, here the indices are strings I mean you can think of them as n bit strings. So, f can be viewed as a function from 0, 1 2 to its 2 to the n to 0, 1, ok. And additionally, if it is a Walsh Hadamard coding you can view that these part you can check that these are linear functions.

So, Walsh Hadamard code words are linear functions, linear functions from 0, 1 to the 2 to the n to 0, 1. And every linear function is an encoding at a Walsh Hadamard encoding of some u, ok. This will look at linear functions in 0, 1 to the 2 to the 2 to the n those can be viewed as encodings of some particular view. So, whether I talk about Walsh Hadamard code words or linear functions they are one and the same those can be easily checked.

So, now going to see some important properties Walsh Hadamard codes; if you are given a Walsh Hadamard code how do we check whether that is indeed a Walsh Hadamard code word, ok. So, we argued that we just need to check linearity, ok.

(Refer Slide Time: 05:07)



So, linear would mean WH code word, ok. So, can we use this to check if a given function or a string. So, when we say that there is a function, we think of it as map from the index set to 0, 1 think of a code, code word which is think of it as a string.

So, let us take find such function and you want to check it is linear or not, ok. So, one way would be to check if f of x plus y is equal to f x plus f y for every x comma y, ok. x can take 2 to the power n different values y can take 2 to the power n different values. So, this would take 2 to the power n times 2 to the power n checks. They are we will have to access those many locations, that those many possibilities, ok. So, that is prohibitively expensive.

But if we check just few positions, we cannot be sure that it is linear, ok. The only way we can be sure that it is a linear function as by checking all the locations in the string. Even if one position is left unchecked there is only one way that that position can be made into a linear function. So, if we and give an incorrect value at that location that no longer be a linear functions. So, we cannot get a deterministic algorithm to check whether the given code word is a linear function, but we can get something which is as good in the sense they will be almost linear, ok. So, we will device a check to see it is reasonably close to a linear function. So, we will call for a definition.

So, we will say that function f and g are rho close, if the probability of f x being equal to g x, when x is chosen randomly from 0, 1 to the power n. If this probability is greater

than rho then we will say that f and g are rho close, ok. They are close by an amount a quantity rho if the probability that this function agree on uniformly sample point with probability greater than or equal to rho, ok.

So, now, what we want to do is really check if a code word is rho close to some linear function. We will see later on at if it is reasonably close to a linear function. Then, from that information we can decide for the exact value of that linear function that any point with very high probability. So, that is going to be the local decoding property of Walsh Hadamard code.

Now, we will see how we can check if a given function is linear, ok. Relate a theorem, this is theorem by Bloom Lubian Luban Field which says that if probability of f of x plus y being equal to f x plus f y, when x comma y are chosen uniformly at random from 0, 1 to the power n, if this probability is greater than some particular amount rho. So, let us say if it is probability is greater than alpha, then f is alpha close to some linear function g. Look at this alpha we required to be greater than half its greater than half, ok.

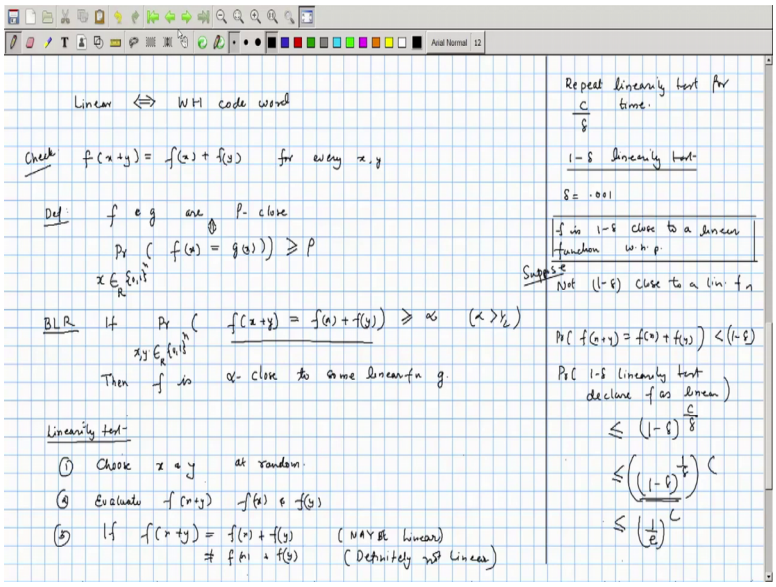
So, this theorem states that if you check for just this property that f x plus y is equal to f x plus f y. If that property is true with high probability greater than alpha then the function is going to be alpha close to some liner function g, ok. So, this means that we can devise a linearity test by just sampling at few locations, ok. In the sense if we run the test of whether f x plus y is equal to f x plus f y for sufficiently many points that will give us a guarantee that the function f is reasonably close to a linear function. So, let us this describe a linearity test, ok.

So, first you will choose x and y uniformly at random, ok. And then evaluate f x plus y, f x and f y, ok. So, this can be directly read from the function. If f x plus y is equal to f x plus f y this does not mean the function as linear, so we will say may be linear. And if this is not equal to f x plus f y and we can say definitely not linear, ok. Now, you can repeat this test some number of time. So, repeat linearity test for say c by delta times because a delta is some parameter. If the linearity after a repeated it many times even if one of the tests out of the c by delta test says that is not linear then the function is not going to be linear, that code word is not going to be a watched smart code word. Otherwise it is going to be when you are going to declare it as, if all the test that passed

then you will say that it is linear. This is known as, so it is known as the one minus delta linearity test, ok.

So, for the time being let us just say that delta equals 0.001, ok. And if this test comes through; that means, f is 1 minus delta close to a linear function with very high probability, ok. We will see why is that the case, ok. So, let us say that it is not 1 minus delta close to a linear function. So, not 1 minus delta close to a linear function. So, suppose this is the case, now BLR theorem says that if it is not close then the probability is going to be less than one minus delta. So, probability that f x plus y not equal to f x plus f y is going to be less than 1 minus delta.

(Refer Slide Time: 13:53)



So, look at BLR Theorem, since the function is not close to some linear function that would mean that, the probability that f x plus y is equal to f x plus f y is not going to be greater than alpha. So, this is not greater than alpha means it is going to be less than alpha. So, alpha here is 1 minus delta, ok. So, this would be in that the probability of the function being incorrectly declared as linear is going to be 1 minus delta from 1 particular run.

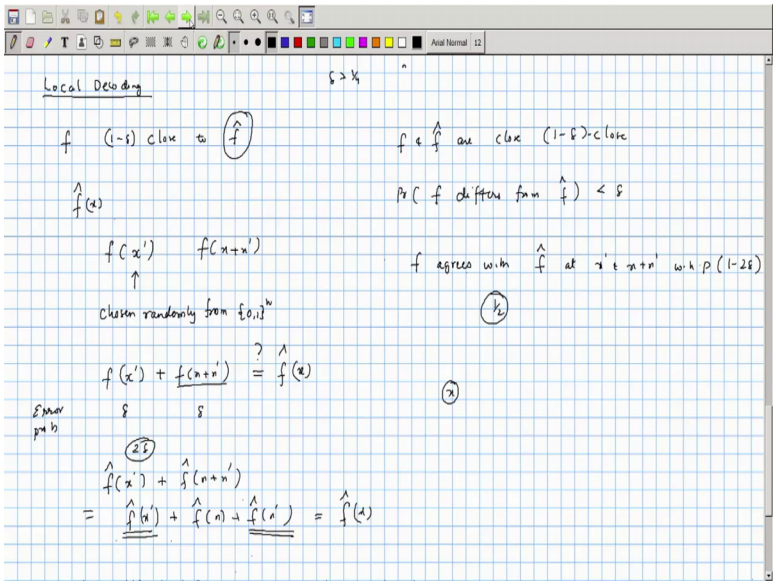
So, if you are running it c by delta times. So, probability that 1 minus delta linearity test declares f as linear is going to be less than 1 minus delta the whole raised to c by delta and this is going to be less than 1 minus delta to the whole power 1 by delta the whole

power c which is going to be less than which is inner quantity is less than 1 by e. So, the whole thing is going to be less than 1 by c, ok.

So, this test called as 1 minus delta linearity test will say that the function is linear with very less probability if the function is actually not linear. You can see that if f was actually linear the test would correctly predict it to be a linear function, ok. If it is not linear if it is not 1 minus delta close to a linear function then the chances of the function being incorrectly declared as a linear function is very small. So, we can detect linearity or we can detect almost linearity, ok.

Now, this is we will see this is sufficient for us. If it is 1 minus delta close to a linear function, but delta let us say reasonably high let 0.001 then we can actually decode that particular linear function. This is called as local decoding, ok.

(Refer Slide Time: 16:05)



So, we were given some particular function this might be a corrupted version of a linear function. So, f we will assume it to be 1 minus delta close to f cap, ok. So, f cap is a correct linear function and we are getting a corrupted version, but these versions are 1 minus delta close, ok, delta let us assume that this is greater than one-fourth. We make it even closer than one-fourth. But it is one-fourth itself would guarantee that there is precisely one function f cap, because this is your function f. There is another function f cap which is at a distance 1 by 4 which means differs in at most one-fourth bits. And there is another g cap which differs an at most one-fourth bit.

Then these both are going to be closed by there will be different at most half the bits, ok. So, since they are different at most half the bits, they and both are linear function that would mean that both of them are equal because each linear function is a Hadamard code word, Hadamard code words verified that two distinct Hadamard and agree at only at most in half the number of bit positions. So, there is a unique f cap to which f is close. And we are interested in knowing the value of f cap at that particular point, ok. And what will help us is the local decoding.

So, if you want to compute f cap x you can essentially use f, ok. What we have to do is compute f x prime, where x prime is some randomly chosen point and compute f x plus x prime. If you choose x prime uniformly at random x plus x prime is also going to be uniformly at random from 0, 1 to the power, it is going to be the random string of length n, although they are dependent, ok.

Now, if we output the value of f of x prime plus f of x plus x prime, we want to know whether this will be equal to f cap x, ok, we will claim that is the case with very high probability, ok. So, we know that f and f cap are close, ok. So, 1 minus delta close. The probability that f differs from f cap at a point which is sampled uniformly at random that is going to be less than delta. They agree with probability greater than 1 minus delta.

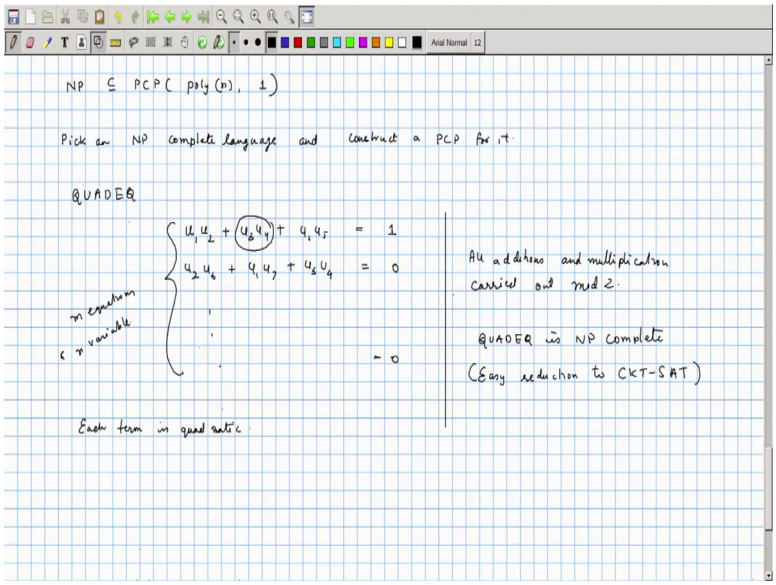
Now, so this value may be different from f hat with probability at most delta. So, let me just write down error probability. This could be incorrect if f evaluated at x prime could be incorrect with probability delta, evaluated at this point could be wrong with again probability delta. Now, so, therefore, the maximum error is only 2 delta. So, these are going to be correct with. So, f agrees with f hat at x prime and x plus x prime with high probability or at least with probability 1 minus 2 delta. So, if delta is greater than one-fourth then this agreement happens, so probability greater than half, ok.

So, in that case f x prime would be f hat x prime plus f hat x plus x prime this is going to be f hat being a linear function, this is going to be f x plus f hat x prime and since all the arithmetic is mod 2, these two quantities cancel out and you will get this as f hat x. So, instead of evaluating the function at a random point x when if you are given a particular point x and you want to evaluate it at that point, instead of directly querying that point you can check at some related points x prime and x plus x prime and based on that you can infer the right end.

So, this will be the correct answer with high probability, otherwise if you are directly querying the adversary knows which is a point at which you have to evaluate and that could have been given the incorrect value. So, here since the adversary has no controlled over which are the query points, unless he makes large number of incorrect values at various query points chance that he will be able to bluff is going to be extremely smart, other words via local decoding you can get the correct answer with high probability.

Since, you are using Walsh Hadamard code what we have gained is first thing we can check if it is linear, second thing is we can do local decoding, ok. You will use these two properties while constructing a PCP for a particular NP complete problem. So, let us just describe NP complete problem that we are going to solve using PCP, ok.

(Refer Slide Time: 22:13)



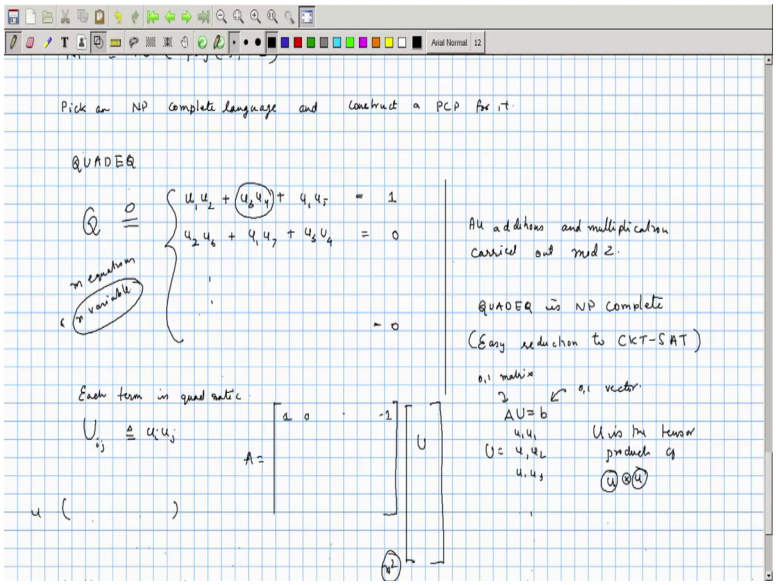
We want to show that NP is contained inside PCP which uses poly n amount of randomness and queries constant number of places, ok. In order to do this what we will do is we will take an NP complete language and construct a PCP for it and the language that we choose is something known as QUADEQ, ok; it is a language of quadratic equations over g f make it clear what that means. So, look at equations of this form. So, u 1, u 2, u k are all variables. So, u 1 u 2 plus u 3 u 4 plus u 1 u 5 is equal to say 1 and u 2 u 6 plus and so on. So, you have m equations and n variables, ok.

And each equation consists of many terms and each terms quadratic term, ok. Quadratic in the sense there are two variables appearing in product form. And a solution for such a

system could be assignment of 0, 1's to the variables in such a way that the equation holds true and all the addition and multiplication are carried out mod 2, ok. You can verify this is NP complete. It is we want to know whether this particular system of quadratic equation has a solution or not. This can, this is a circuit sat can be reduced to this, ok. So, if you can solve this you can solve any other NP complete problem.

There is an easy reduction to circuit sat, you can we are not going to look at that reduction, ok. So, this is a particular problem that we are looking at how do we solve this using PCP. We will write this equation in a slightly different format. So, this has n variables and m equations. We can write this entire system of equations as a matrix and each u i u j think of that as a new variables.

(Refer Slide Time: 25:15)



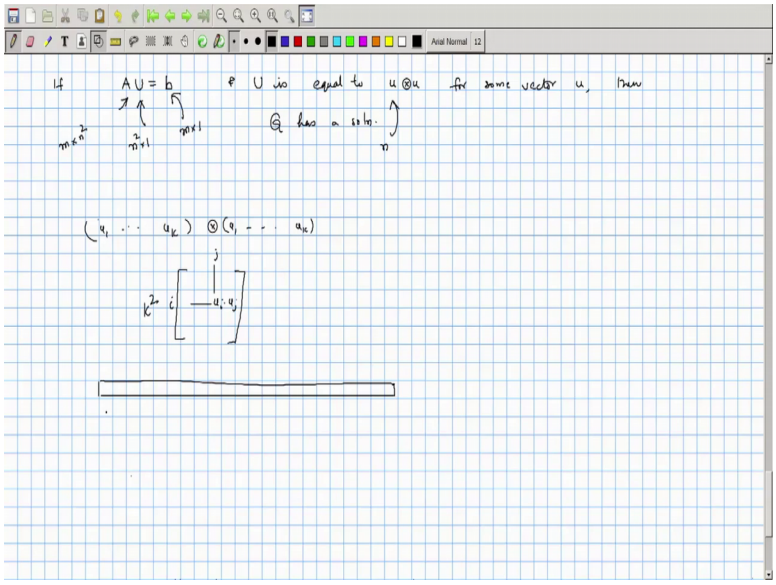
Let us have a new variable capital U i j, and that is defined as u i u j, ok. So, now, with this notation what we have is we can write this entire thing as a matrix, ok. And these entries these this matrix which we call as A, the entries in this are going to be 0, 1 entries, because all multiplications and additions mod 2, ok. And this is going to be multiplied with the vector U, and U consists of all the capital U i js.

Now, if we solve that that does not mean that the original system of equation which let us call it as Q. So, Q may not have a solution, ok. They will not solve this equation AU equals b, where b is the vector consisting of all the RHS, ok. So, A is a 0, 1 matrix and b is a 0, 1 column vector, and U consists of u 1 u 2, u 1 u 3, may be u 1 u 1 and so on, ok

so that n square entries in U, ok. So, if that has a solution then we want to say that QUADEQ has a solution, ok but there is some dependency between these u i, u j's.

So, if additionally, if you that U is the tensor product of u, where u is a solution and if you think of the quadratic equation in that the solution is going to be a vector on n variables, ok. So, if you call that n vectors, I mean if you call that particular vector as u. So, this is the solution to the quadratic equations, and if your capital U is a tensor product of u and u, ok, then we can say that the QUADEQ has a solution, ok. So, let us this write this down cleanly.

(Refer Slide Time: 27:41)



If AU is equal to b and U is equal to u tenser u for some vector small u over 0, 1 then Q has a solution, this is when if and only if condition, ok. So, here A is going to be m cross n square matrix, U is going to be an n square cross 1 matrix, and b is going to be a m cross 1 matrix and small u is going to be an n matrix, ok.

Now, what is this tensor product? So, if you take a vector u 1 up to u k, and if you tensor product it with u 1, u 2, u k you get a matrix which is a k square matrix where the i jth entry is going to be u i into u j. So, these conditions you can check that if a b is equal AU equals b and u is equal to the tensor product of some vector in which the components are chosen from 0 and 1, then Q has a solution then the reverse also true, ok. So, what we will try to show is that the solution to this particular system of equations can be

converted into PCP, ok. So, you can construct some particular PCP that encodes the solution of this particular QUADEQ that is what we will do in the next session.