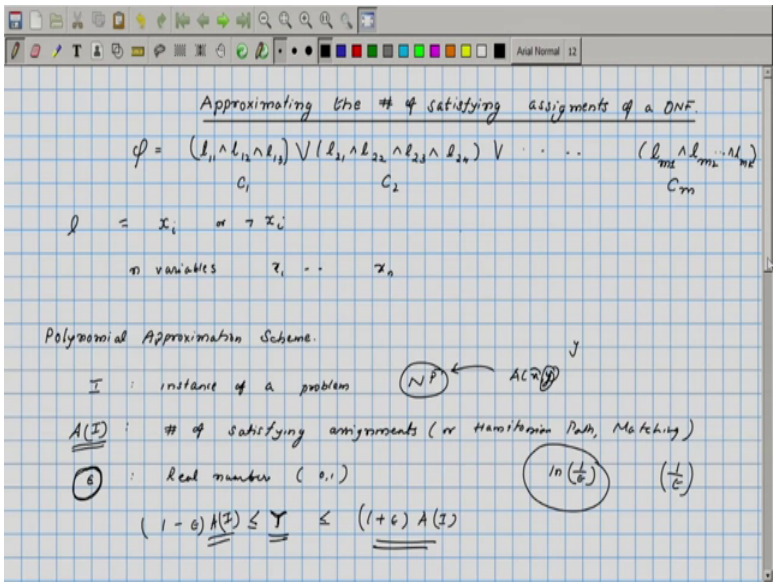
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 30
DNF counting

In today's lecture we will see how we can approximate the number of satisfying assignments for a DNF formula. So, the DNF formula is a formula is Disjunctive Normal Form which means you have clauses C 1 C 2 to C m and each of these clauses is a disjunction of literals.

(Refer Slide Time: 00:41)



So, each formula is of each clauses are the from l 1 and l 2 and l 3. So, on where each a l i is a literal; that means, it is a formula it is a of the form x i or not x i. So, let us say l any general literal is of the form its x i or not x i for some i where x i is the variable. Now, what we are interested in is counting the number of assignments. So, suppose there were n variables x 1 to x n there are 2 raise to n different assignments possible, we need to count the number of assignments which are satisfying this particular DNF in other words the number of assignments which causes the formula to evaluate to 1 ok.

So, first of all we need to understand what is the meaning of approximating in this context. The first notion that we will look at is polynomial approximation scheme. So, suppose I is an instance of a problem; that means, one particular formula and we are

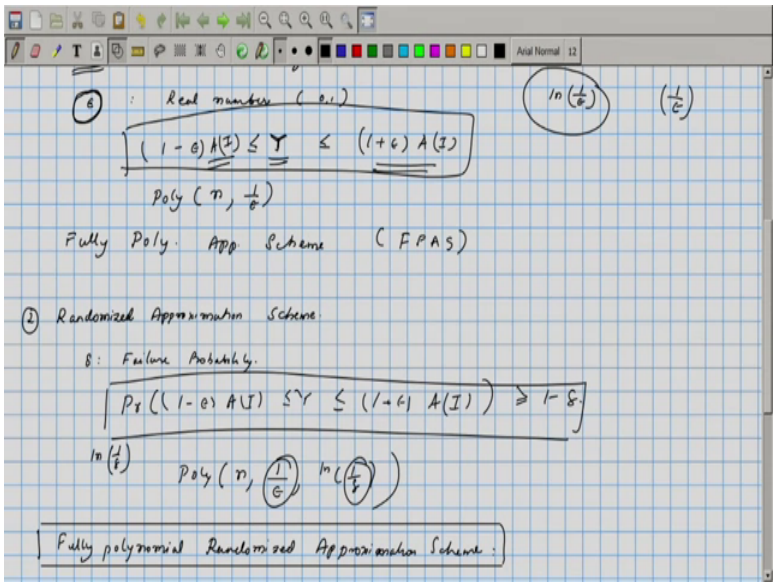
interested in A I that is. So, here you could think of it as number of satisfying assignments or let us say Hamiltonian path matching etcetera; number of Hamiltonian paths in a given graph a number of matches in the given graph. So, A I denotes the number of assignments which can satisfy the particular problem or some parameter associated with each of these instance. For each of these instance if you think of it as an a problem belonging to the class non-deterministically NP, then you can look at A I as the number of certificates or number of y's which will satisfy the formula corresponding to the non-deterministic problem.

So, we often write it as A x y we write a predicate associated with each NP problem instance A x y where y is what we call as a certificate. So, A I you can look at as the number of certificates and when we say that we have a polynomial approximation scheme; that means, for any instance of the problem and for a given number epsilon. So, epsilon we can think of it is a real number, let us say between 0 comma 1; we can output something let us call it as Y this lies between. So, the actual value was A I, then our output will lie between 1 minus epsilon times A I and 1 plus epsilon times A I.

So, if by A I if we denote the number of satisfying assignments and your Y the output of your algorithm must lie between 1 minus epsilon times the actual count and 1 plus epsilon times actually count. And, if you can have such an algorithm the time taken should be polynomial in the input size and there is an epsilon here. So, if you look at epsilon as a number it will take let us say log 1 by epsilon we are taking 1 by because it is a number less than between 0 and 1. So, log 1 by epsilon is the amount of space required to present this number or the input size is log 1 by epsilon.

So, we could have said that the algorithm should run in polynomial time it being polynomial both in the input size and in log 1 by epsilon, but in this case we actually allow a little more leeway say happy if it runs in poly 1 over epsilon.

(Refer Slide Time: 05:49)



So, when we say polynomial approximation scheme with approximation as let us say I mean epsilon we will say that it has to run in poly, if you think of the input size as n then should be poly n and 1 by epsilon. And if we could find such an algorithm for every possible epsilon then we will say that is a Fully Polynomial Approximation Scheme and we call it as FPAS. Because that is one notion of approximation here we insist that the algorithm has to be deterministic.

So, deterministically an output has to be generated which lies very close to the actual answer, it deviates with the actual answer by an amount epsilon. This is the first notion regarding approximation. The second notion that we will have is we will allow our answers to be let us say randomized. So, we will think of randomized approximation schemes. So, everything remains pretty much same as polynomial approximation scheme, but we will insist that this output which should lie between these they need not always lie there, but it lies with a certain probability delta.

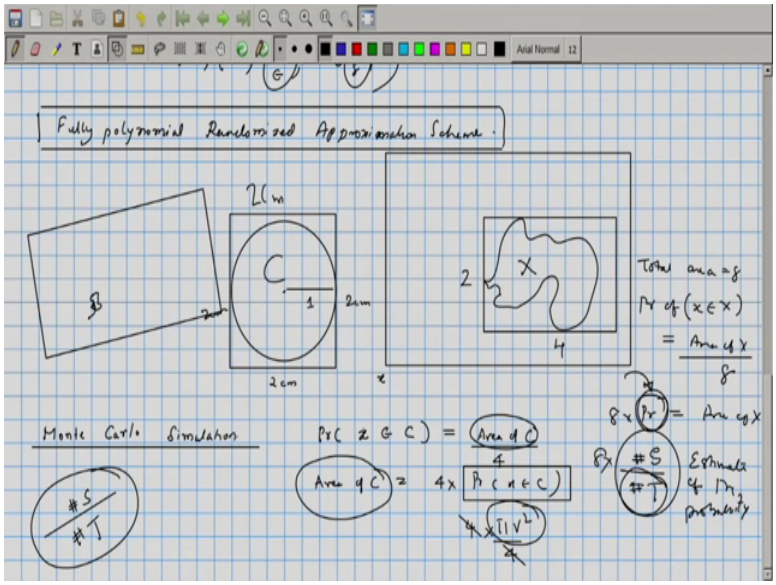
So, the additional parameter is delta if we say that this is the failure probability when there is the chance that the algorithm fails. So, we will say that the probability that the output lies between 1 minus epsilon times A I and 1 plus epsilon times A I this is going to be greater than 1 minus delta; means the success probability is at least 1 minus delta. And if we could do this for every delta, delta again being a real number between 0 and 1, the input size is log 1 by delta. So, we will insist that the algorithm runs in poly in

polynomial time polynomial n; polynomial n let us say 1 minus epsilon and log 1 by delta. And, if we could do this for every epsilon and every delta then we will say that we have a fully polynomial randomized approximation scheme.

So, this would be, so, when we look at any approximation problem this would be the best case scenario that is if we can find an algorithm which is which satisfies these conditions. And, if we are unable to do that we will try and see if we can do it for just mean to say certain epsilons, but delta we still want the failure probability we want it to be arbitrary low that is for any failure probability we should essentially have a an algorithm. And its running time should be bounded by polynomial in log 1 by delta, but regarding the approximation factor where a little less stringent ok.

So, this is what we mean when we say we have an approximation algorithm a fully polynomial randomized approximation scheme for a particular problem. Now, let us keep this definition inside for a minute and think about a simple problem.

(Refer Slide Time: 10:03)



So, let us say we have a circle whose radius is 1 ok, we need to estimate the area of this could have been any other geometric figure how do we estimate the area? So, there is a technique called as Monte Carlo simulation ok. What does the basic idea? Fit this circle inside the square ok. So, this is a square of size 2 centimeter. So, if we randomly choose a point from the square the probability that. So, let us call the randomly chosen point let

us say x, probability that x belongs to C is equal to area of C divided by the area of this square which is 4 ok.

Now, this will also mean that area of C is equal to 4 into probability that x belongs to C ok. So, if we could estimate this probability exactly that would mean that we can compute the area of c. So, now, probability that x belongs to C is going to be nothing, but pi r square by 4 and this multiplied by 4 gives us pi r square in some sense we already knew this ok. So, we just plugged in that to compute the probability, but let us say it is a more irregular figure ok.

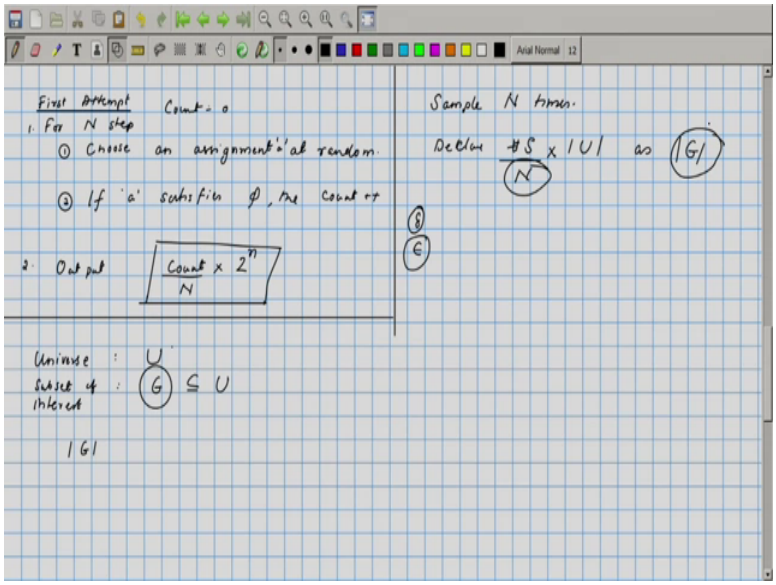
So, we enclose it in a in a known area let us say this is 2 and this is 4, the total area is 8 probability of x belonging to let us say x that region let us call it as x is equal to area of x divided by 8. So, 8 times that probability is going to be area of x. Now, how do we estimate this probability? One thing we can do is we can just repeatedly toss ok. So, let us say independently we toss the 100 coins and out of these 100 points some of them will lie inside and someone they may lie outside. Should have a method to figure out if the randomly chosen point lies inside or outside the region of interest. If we could do that the number of points which falls inside let us say that is success.

So, number of success by number of trials this is a good estimate of the probability you can argue that no more theoretical fashion, but you can intuitively feel that the fraction of points which lie inside is a good indicator of the probability that the point chosen lies inside ok, 8 multiplied by that gives us the area. So, that is a way to compute the area this is this same method can be extended to counting or approximate counting. So, this gives an approximate area if we could compute the probability exactly then the area computations would have been perfect it would have been exact, but in we cannot compute the probability exactly since we are unable to do that we estimate it and in the estimation there can be errors.

The estimation by central limit theorem certainly is going to be better and better as the number of trials becomes very large. Also you can see that if we had chosen a much larger area to contain this particular area of interest then the number of trials will have to increase. If you were to get let us a reasonable approximation to increase the accuracy it is best that the area which is holding this particular region is as close to the actual area as possible ok.

So, that will give better and better bounds, on the number of trials that we have to do if this was a tiny region inside a large let us say box then we do not expect to hit this region too often. But the quality of the answer that is if you look at the number of success that is points which are chosen and are lying inside the area of interest divided by the total number of trials. This answers accuracy depends on the number of trials in the number of trials will have to be larger and larger if this area was large in comparison to the area of interest ok.

(Refer Slide Time: 15:33)



So, with these insights we will try and have an approximation algorithm for DNF. So, first attempt this will be our algorithm choose an assignment at random to do this for n steps if. So, let us call this assignment a if a satisfies phi then a count plus plus. So, initially we will have count equals 0 and then the last step would be number of. So, we are interested in the count. So, we will output its a count divided by n this is a fraction of satisfying assignments multiplied by total number of assignments we had 2 raise to n assignments ok.

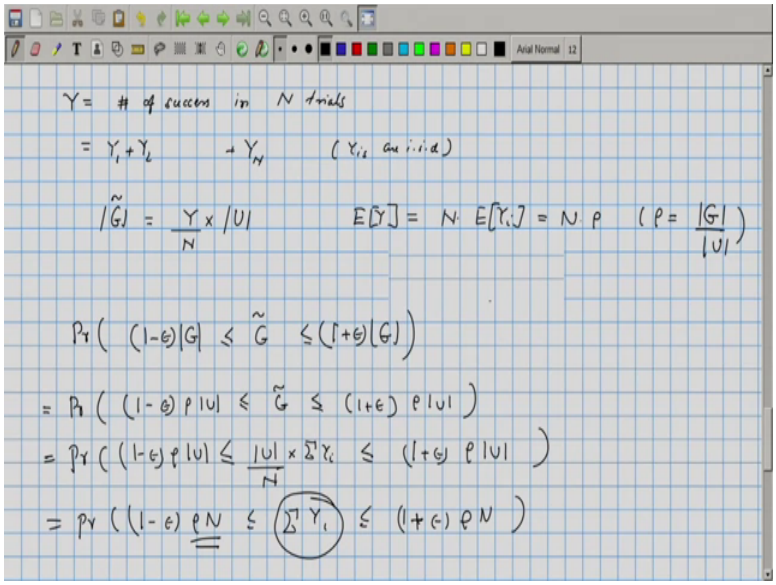
So, we can round this off to an integer value and that is going to be our output. This is the algorithm any good what are the probability, what are the issues with this what are the I mean what can we say about the success probability and the quality of answer. So, let us look at a more general problem; let us say we had a universe which we call as U and let us say we have the set of interest which we call as G.

So, G is a subset of U, we are interested in computing size G ok. So, let us call this U size of U to be. So, this is ok. So, now, if we follow a similar algorithm Monte Carlo algorithm to compute the a the size of G then we would have sampled n times. So, the algorithm is simple sample n times declare number of success divided by N times size of U as let us say size of G ok.

So, when we do this what all could go wrong. How many times will we have to run the algorithm n is what we need to determine in order to get a reasonable estimate. So, suppose we say that we want let us say an algorithm which has error probability of at most delta and let us say approximation factor as epsilon.

Then how many times will we have to run this algorithm in order to get these parameters. The problem that could be there is G could be first of all very large or very small if G is very small compared to the size of the universe then we expect that this is not going to be a good algorithm and we will see the quantitative dependency between these.

(Refer Slide Time: 19:53)



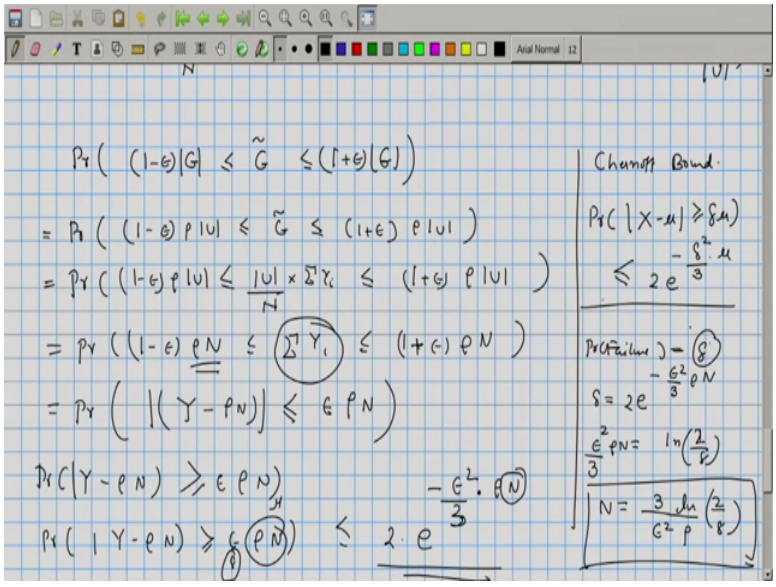
So, we will define few random variables Y is equal to number of success in n trials. So, it could be any number between 0 to n and we can think of this is Y 1 plus Y 2 plus Y N where each these are i i d random variables ok. The answer that we would have declared is. So, size of G we will declare it to be equal to Y by N into size of U ok, estimate of G that we will give is Y by N times U we want to know. So, note that we can calculate expectation of Y that is going to be equal to N times expectation of each Y i and each Y i

we can say that its expectation is rho, where rho is equal to say G by U size of G by size of U

The probability that let us call the estimate as G tilde; so, G tilde lies between 1 minus epsilon times G and 1 plus epsilon times G, this is a probability that we need to estimate ok. And then we want this probability to be let us say greater than 1 minus delta; that means, the failure probability is going to be delta and for that what should be the N that is what we will compute.

So, this probability it is going to be equal to probability that 1 minus epsilon and size G is going to be rho into U and this is equal to probability that 1 minus epsilon times rho U lies between G tilde is nothing, but U divided by N into summation Y i. So this probability is nothing, but the random variable Y which is summation of Yi's lies between 1 minus epsilon times rho N and 1 plus epsilon times rho N and rho N is the expectation of this random variable. So, we can simply write this as.

(Refer Slide Time: 23:07)



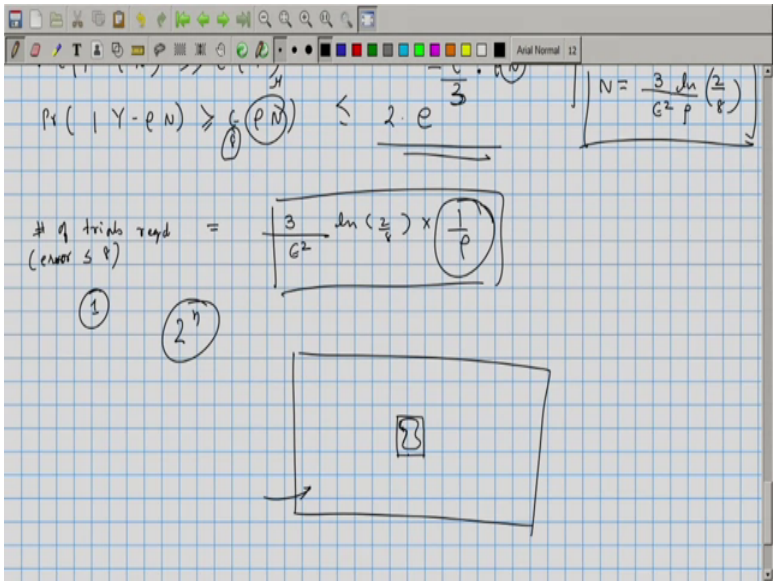
The probability that Y minus expectation of Y rho N the modulus of that, is less than epsilon times rho N. So, we can straight away apply Chernoff bound ok, Chernoff bound applies to the complement of this event. So, let us if you look at the complimentary event so, probability that Y minus rho N the absolute value is greater than or equal to epsilon rho N.

Chernoff bound says if you sample Yi's independently with rho then there some x will deviate from its expectation by an amount more than delta U is going to be less than 2 e raise to minus say delta square by 3 times mu ok. So, this is the form of Chernoff bound that we can just straightaway use here and that will give us the probability that Y minus rho N is greater than epsilon times rho times N.

So, this is mu and this is going to be delta. So, this is going to be less than 2 into e raised to minus say epsilon square by 2 into sorry epsilon square by 3 into your mu is going to be rho N ok; so, that is that is what we obtained. So, we will look at the failure probability the failure probability is less than this. So, if we can tolerate the phase the probability of failure if it is equal to delta at most delta and we can equate these two quantities to determine what should N be ok. So, equating those if you said delta this is a failure probability this is equal to 2 e raise to minus epsilon square by 3 rho N, we can just take log.

So, epsilon cube by 3 rho N is equal to log 2 by delta. So, N will be equal to 3 sorry by this is epsilon square 3 by epsilon square log 2 by delta by rho. So, the number of trials that you have to do to get a probability of failure no more than delta is going to be this much.

(Refer Slide Time: 26:19)



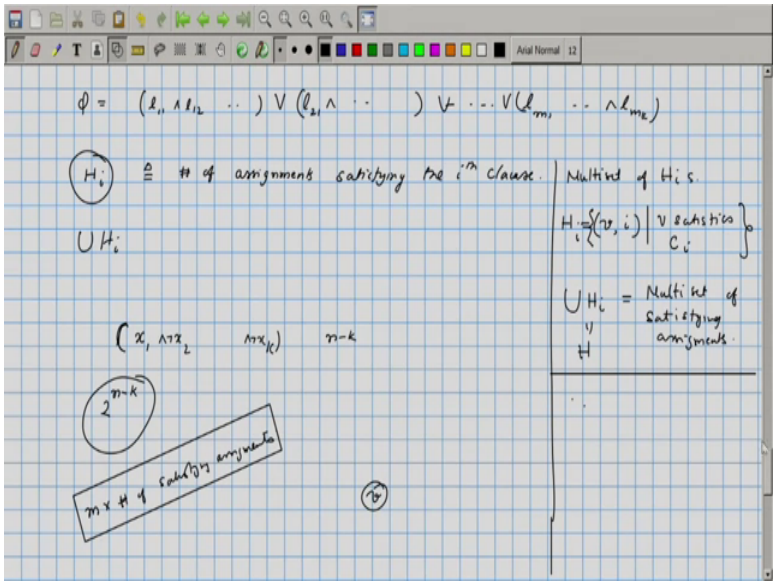
So, number of trials required will be equal to 3 by. So, this is for error less than delta this is 3 by epsilon square log 2 by delta into 1 by rho. So, if this rho was very small, rho was

the fraction of inputs that we had tried which were good, rho is size of G by size of U, G was the set that we were interested in and U was the universe. If rho was very small then this is not a good algorithm. So, in DNF there could be a DNF whose there is just 1 satisfying assignment amongst let us say 2 raise to n possible assignments and then the number of trials record could be large ok.

So, that is the reason why we cannot use the simple Monte Carlo algorithm to solve the approximation problem for DNF counting; so, what is the way out. We were now sampling from here should not expect that this is going to be a good sampling because they said that we had was very tiny compared to the region from where we are sampling. If you could find something which is small enough to encompass this then we could do a better approximation and that is what we will see in the next part ok.

So, instead of sampling from the set of all possible assignments we will try and sample from a small region ok. So, what does these regions going to be ok. So, let us look at this structure of DNF formulas little more carefully.

(Refer Slide Time: 28:29)



So, if the formula was of the form let us say l 1 or l 1 1 and l 1 2 so on or l 2 1 and so on or l m 1 and l m k. Can we try and count how many satisfying assignments can be there they will split into parts. So, let H i denote the number of assignments satisfying the ith clause ok.

And what we are really interested in this union of H i, can we determine any of these if we find the H i and we could maybe try inclusion exclusion formula, but since there are m clauses the inclusion exclusion formula would essentially have 2 raise to m different terms. So, we cannot really go by that root that is going to be computationally very costly.

So, what can we really do here. Let us see if we can just compute each H i. If you look at each H i they are of the form let us say x 1 and x 2 and x k maybe not and. So, how many satisfying assignments are there for this? Well, there is only one possible value of these variables that will satisfy the particular clauses H i, but the other n minus k variables can be arbitrarily assigned.

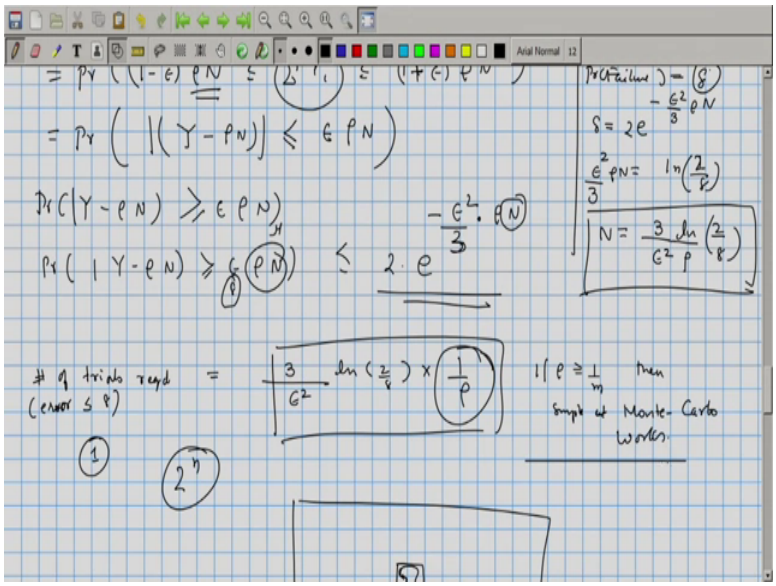
So, each assigned count is extremely simple, the number of satisfying assignments for the ith clause can be simply computed as 2 raised to n minus k, where k is the number of variables that is appearing in the or number of literals appearing in that particular clause. Note that if any clause appears more than once we just throw them away that simplification can be done some variable appears both as positive and negative then of course, that clause is not satisfiable therefore, we can just throw it away.

So, 2 raise to n minus k under these assumption 2 raise to n minus k is the number of assignments which satisfies the ith clause. If we wanted to take 2 of them simultaneously then it becomes little more complicated, but that also you can compute the formula. But the real issue here is we cannot compute all of them there are too many of them 2 raise to m of them. So, we cannot apply inclusion exclusion.

What we will do is instead of sampling from the entire when we run our Monte Carlo simulation or the Monte Carlo algorithm instead of sampling from 2 raise to n possible assignments. We will sample from another sample space which whose size is not too large it is at most m times the number of satisfying assignments ok.

So, if we could find a space which contains at most m times the number of satisfying assignments then by this method rho is going to be at most m sorry 1 by m so, 1 by row is m. So, that is going to work in 3 by epsilon square log 2 by delta ok.

(Refer Slide Time: 32:35)

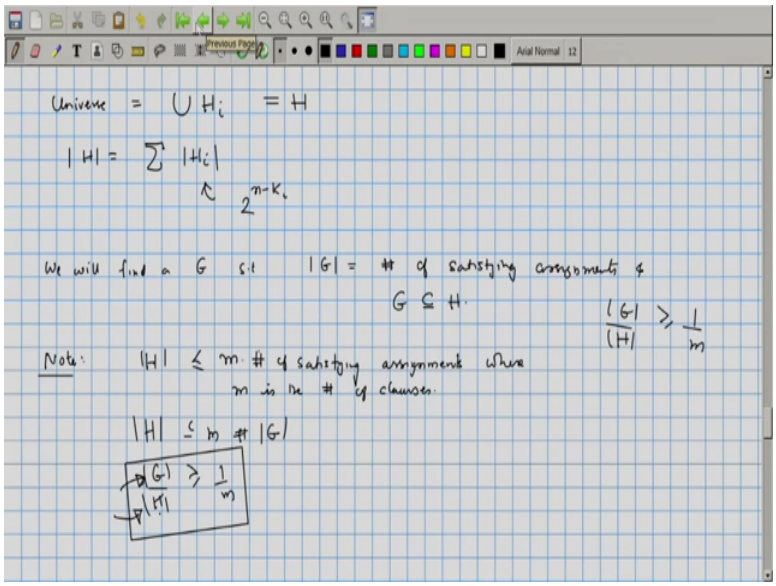


So, if rho is equal to 1 by m or if it is it is a greater than or equal to 1 by m then simple Monte Carlo works. So, that sample space is what we will construct its a simple sample space to construct. Let us imagine the multi set of a Hi's. So, h i here we denoted it as a number we can overload it and say that H i is a set containing all the assignments which satisfy the ith clause.

So, when you say multi set we will look at the assignment comma i ok. So, consider the pair v comma i ok. So, let us say we will again H i itself H i is equal to the set of all v comma i such that v satisfies the clause i ok. So, clearly if you take union of Hi s that is going to be multi set of you can put it i mean in 1 to 1 correspondence with the multi set of satisfying assignments that is for every satisfying assignment we are counting the number with its multiplicities.

If the assignment v satisfies both let us say clause 5 and 3 and 25 it is counted 3 times, this is once for each clause it satisfies. Now we could define another set which is basically so, this is going to be our universe. So, union H i have which we will call as H.

(Refer Slide Time: 34:45)

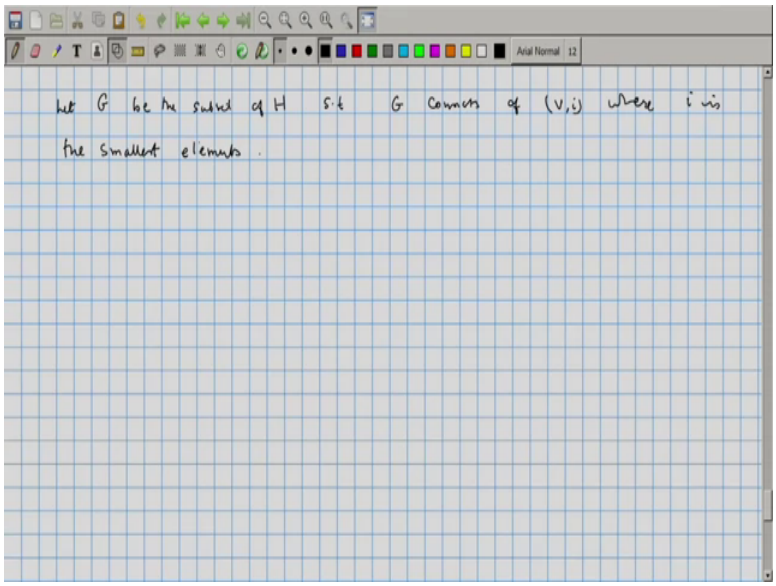


So, universe is going to be union H i which we will call as H ok. Now in this universe when the count means obtaining the count of this universe is simple because that is just going to be sum over sizes of H i s because these each H i has a unique i in the second coordinate. So, you look at each H i the elements inside that H i s the first component is an assignment and the second component is a number indicating i. So, it is different for different classes. So, this summation is just 2 raise to n minus k i where k i is the number of literals appearing there. So, size H i can be easily computed.

Now, we will define G to be the subset. So, find G such that size G is equal to number of satisfying assignments and additionally G is a subset of H ok. Now number of satisfying assignments and H what is the relationship? So, H is not going to be so, it surely will be less than m times number of satisfying assignments where m is the number of clauses. This is because you take any satisfying assignment it can be present in at most m clauses ok.

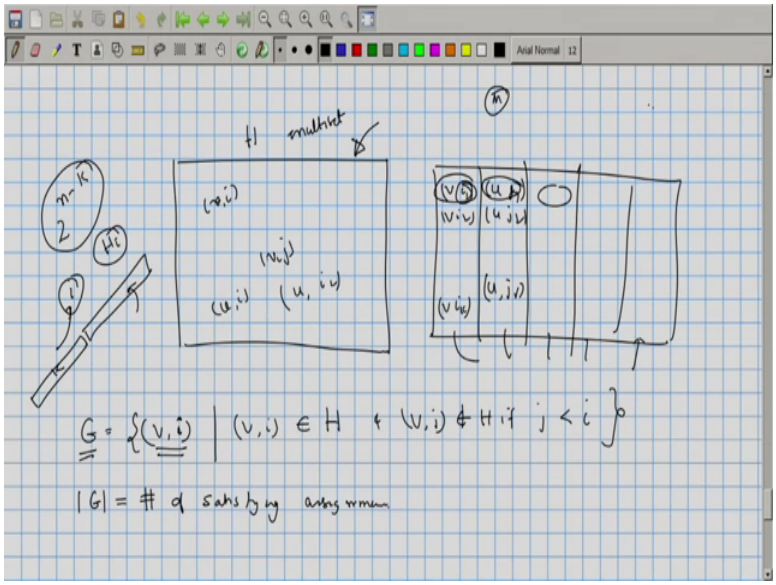
Therefore each quantity can each assignment can appear at most m times therefore, total number the size the multiset is bounded by m times the number of assignments. So, when you get these the set G which is equal to number of satisfying assignments and this is the subset of H size of G by size of H is going to be greater than 1 by m ok. So, G by H is going to be greater than 1 over m and that will take care of all the requirements ok. Now how do we construct this set? It is simple.

(Refer Slide Time: 37:45)



So, let G be the subset of H such that G consists of elements of the form v i where i is the smallest element may be i will just rewrite it.

(Refer Slide Time: 38:47)



So, we have this set H which is the multi set. So, this set these various elements v i v j u i u i 2 etcetera. Now we can think of this as split into various assignments. So, let us say v i 1 v i 2 v i k. So, v is an assignment which appears k times and then let us say u appears say j 1 u j 2 u j r and u appears r times and so on. So, we want to pick one element each

of these for that we will pick this smallest j r ok. So, G will basically be the set of v i such that v i belongs to H and v j does not belong to H if j is less than i ok.

So, pick the smallest elements on the basis of the second coordinate for each assignment. So, we are the number of elements that we pick is going to be equal to the number of satisfying assignments. So, this is easy G is equal to number of satisfying assignments ok. Now, how do we check if a particularly now let us sample one of the elements and we sample uniformly from H. Well, we can I mean the moment of thought would tell you that from this multi set if you wanted to sample that is not a difficult thing because you can first choose a particular clause uniformly at random and then from that.

So, if you choose let us say this the second coordinated random. For that particular second coordinate, all the satisfying assignments of that there are 2 raise to n minus k of them. They can be found by fixing the first k coordinates first k here means whatever of the variables it is appearing in that particular clause those who set to the value as prescribed by the clause. And for the other variables you just choose randomly ok; uniformly at random toss a coin and fix the values of the other variables and that will give you a uniform sampling from each of these H i's and that basically translates into a uniform sampling for the entire set.

So, you can uniformly sample from H. Further once you have picked one particular element, how do you check whether that belongs to the set G. Well if your i was some particular element, for every element smaller than i, there are at most m of them you can check whether they belong to any of these smaller elements. There are only m of them. So, you can actually do this computation that computation polynomial time.

So, this verification of whether a particular assignment comma i pair belongs to G can be done in polynomial time and therefore, we are done with the requirements we could sample from this set x uniformly at random. The size of H is known and we can in polynomial time figure out whether the sampled element belongs to G or not and therefore, that will give us a polynomial time approximation scheme for computing the number of satisfying assignment of DNF formula.

We will stop here and we will continue with other approximation algorithms in the next class.