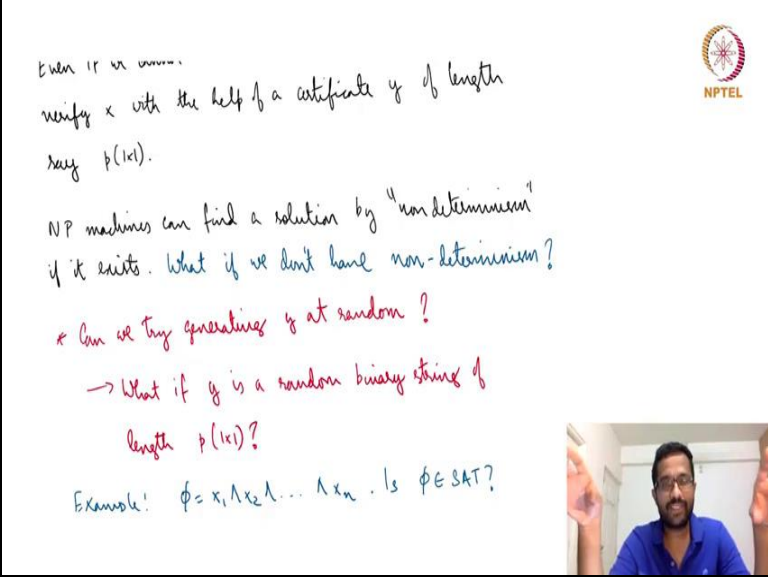


**Computational Complexity**  
**Prof. Subrahmanyam Kalyanasundaram**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**

**Lecture -27**  
**Randomized Complexity Classes Part 1**

(Refer Slide Time: 00:15)




Even if we cannot  
verify  $x$  with the help of a certificate  $y$  of length  
size  $p(|x|)$ .


NP machines can find a solution by "non-determinism"  
if it exists. What if we don't have non-determinism?

\* Can we try generators  $y$  at random?

→ What if  $y$  is a random binary string of  
length  $p(|x|)$ ?

Example!  $\phi = x_1 \wedge x_2 \wedge \dots \wedge x_n$ . Is  $\phi \in \text{SAT}$ ?





Hello and welcome to lecture 27 of the course computational complexity. So, today we are going to start talking about randomized complexity classes all. So, we will first motivate why it makes sense to have a randomized algorithms and then we will see complexity classes that are derived out of randomized algorithms and we will see like around 3 or 4 of them in the in the next couple of lectures.

So, before getting into randomized complexity classes let us try to see what is the class P. A language  $L$  is in P a language  $L$  is in P if there is a polynomial time decider for the for that language meaning whenever  $x$  is in the language the decider accepts whenever  $x$  is not in the language that is decider rejects  $x$ . So, here in this I have used the notation  $A$  for the decider and the decider is a deterministic polynomial time decider.

What is NP we had NP is a class of languages that have a non-deterministic polynomial time decider. However we saw a verifier based characterization what is the verifier based

characterization  $L$  is a not NP is a class of languages  $L$  which have polynomial time verifiers meaning whenever  $x$  is in the language there is a certificate string  $y$  which which can be used to verify the membership of  $x$ .

So,  $y$  2 properties  $y$  is not too long  $y$  is length of  $y$  is at most polynomial in the length of  $x$  and 2 the running time of the verifier is polynomial in the length of  $x$  and also in the length of  $y$  but then anyway the length of  $y$  is polynomial in the length of  $x$ . So, in this notation here we use  $V$  to denote the verifier algorithm. So, we have the 2 classes  $P$  and  $NP$ ,  $P$  has polynomial time deciders  $NP$  has polynomial time verifiers.

Now again  $NP$  is a class of languages that can that can be efficiently verified uh. So, which means there is a certificate or a proof or a witness these are the 3 terms that we use to describe the string  $y$ . And once the  $y$  is there it can be it can be very used to verify the membership of  $x$ . So, for instance if the problem is testing whether the graph is 3 colourable the certificate is the actual 3 colouring the or one of the certificate is the actual 3 colouring.

So, then you can use a 3 colouring to verify that the graph is recolourable if the graph is not 3 colourable whatever you try it will not work none of them will be proper colouring. Anyway that is a very very brief introduction to  $P$  and  $NP$  now let us see what is; so,  $NP$  machines so, there are different ways to think of  $NP$  machines. So, some may say that so, what we said in this lecture is that if there is a correct solution if there is a correct certificate somehow the machine magically finds this solution it is that is the power of non-determinism.

I have also seen other sources where they say that you can think of it as being able to parallelly test all the strings all the certificates or all the computation paths can be parallelly tested. So, these are all different ways to think of non-determination. So, I personally do not do not agree with this parallel notion because that I think leads to more confusion I would like to think of it as the machine automatically finds the path using the power of non-determinism.

**(Refer Slide Time: 04:26)**

Example:  $\phi = x_1 \wedge x_2 \wedge \dots \wedge x_n$ . Is  $\phi \in \text{SAT}$ ?  
 Obviously satisfiable. Just set  $x_1 = x_2 = \dots = x_n = \text{TRUE}$ .  
 Consider an algorithm that generates an assignment at random. If  $\phi$  evaluates to TRUE, alg. concludes that  $\phi \in \text{SAT}$ . If  $\phi$  evaluates to FALSE, alg. concludes  $\phi \notin \text{SAT}$ .  
 ... repeat the above algorithm  $n$  times,

So, the question is now what if we do not have non-determinism or not. It is well established now that non-determinism is some kind of a superpower what if we do not have non-determinism. So, one possibility is now so, let us motivate randomness. So, can we not guess the witness string at random. So, instead of guessing using non-determinism what if we use randomness. Can we generate the witness string at random then what would happen.

So, what if  $y$  is a run like if we know this the length of the witness string or expected length of witness string can we not generate such a thing at random using randomness. So, let us try to see some problem and let us see what happens. So, let us see a satisfiability instance  $\phi$  is equal to  $x_1$  and  $x_2$  and so on till  $x_n$  variables is this satisfiable or not. Again this is an easy question in this sense. So, it is an end of  $n$  variables obviously it is satisfiable all.

So, it is obviously satisfiable just SAT all of the variables to true. So, the  $x_n$  is also true it is an and function obviously however this is something that is easy for us to see when we when we look at the formula in a computer or a turing machine does not have the have this kind of a judging capability it just gets something and it has to do a procedure that is generic. So, it cannot it will not be able to look at this and say yes this is the and function and so, we can is obviously satisfiable.

So, now what will a computer try to do? Computer like if it if it has to generate the satisfying. So, what it may try to do here in this thing in this model about generating the witness or certificates at random. It will try to generate assignments the true false assignments at random. So, then what is the probability of it succeeding. So, consider an algorithm that generates an assignment at random.

So, the algorithm is very simple it generates an assignment at random. And let us say the assignment is; and then if phi evaluates to true algorithm concludes that phi is in SAT if phi evaluates to false algorithm concludes phi is not in SAT. So, the algorithm has to decide whether it is satisfiable or not. So, it generates a random assignment it evaluates phi. So, now let us see what happens in this case?

**(Refer Slide Time: 08:45)**

Consider an algorithm that generates an assignment at random. If  $\phi$  evaluates to TRUE, alg. concludes that  $\phi \in \text{SAT}$ . If  $\phi$  evaluates to FALSE, alg. concludes  $\phi \notin \text{SAT}$ .

$$P[\text{correct (for the above } \phi)] = \frac{1}{2^n}$$

$$P[\text{error " "}] = \frac{2^n - 1}{2^n} = 1 - \frac{1}{2^n}$$

If we repeat the above algorithm  $n$  times,

$$P[\text{error}] = [1 - \frac{1}{2^n}]^n \approx e^{-\frac{1}{2}}$$

$\therefore [1 - \frac{1}{2}]^n \approx \frac{1}{2}$

NPTEL

You may be able to guess now what is likely to happen. So, the formula is just an and formula how many assignments are there the number of assignments is there are  $n$  variables  $2$  power  $n$  assignments are possible how many of them. So, phi as we saw is obviously satisfiable how many of these lead to. So, that we want the algorithm to conclude that phi is satisfiable how many of the randomly generated assignments lead to the conclusion that phi is in SAT.

There is only one assignment which is setting  $x_1, x_2, x_3$  everything to true that leads us to concluding that phi is in SAT. So there is only one out of  $2$  power  $n$  assignments that will lead us

to conclude that  $\phi$  is SAT for all the other  $2^{n-1}$  assignments we will conclude that  $\phi$  is not in SAT. So, the probability of success at least for the above formula for the above  $\phi$  is equal to  $1/2^n$  and probability of error again for the above  $\phi$  maybe I will just write that down is equal to  $2^{n-1}$  divided by  $2^n$  or  $1 - 1/2^n$ .

So, you see how bad this is this particular algorithm you what we try to do is we have this is  $x$  it is it is succeeding but with a very very small probability  $1/2^n$ . So, it is one by exponential probability inverse exponential probability which is very very small and the rest of the time it is going to the it is making an error it is inferring the wrong conclusion. So, this is an example of a randomized algorithm. So, now things may seem hopeless depressing and all that but actually this is just a starting point and we will see how to improve from here.

And believe it or not randomized algorithms have a are usually extremely nice to work with many times the algorithms are extremely simple to describe. So, this algorithm for whatever it is worth it was even though the probability of success is not that good it was very simple I just you just randomly generate an assignment you plug it in and see whether it works or not. So, usually randomized algorithms are extremely simple to describe you do not need much to it you do not need it is not.

So, difficult to understand it is not. So, difficult to program it is it is very easy to program and usually they perform decent reasonably well. So, again this is just an exception but it was this is just an exception and it is deliberately chosen. So, that I can see how I can show you how to how this can be led to an improvement. So, you should there is a lot of advantage there are a lot of advantages to randomized algorithms.

And we will see again this is not a course on algorithms or randomized algorithms. So, I will not go into the benefits of that rather we will focus on the complexity classes derived from randomized algorithms. Now let us see what we can do. So, obviously this probability of success is poor.

**(Refer Slide Time: 12:04)**

NPTEL

Boosting  
 If we repeat the above algorithm  $n$  times, in  $n$  trials.


Prob [ error ] =  $[1 - 2^{-n}]^A \approx e^{-\frac{A}{2^n}}$

False negative  $\lim_{x \rightarrow \infty} [1 - \frac{1}{x}]^x = \frac{1}{e}$

Can we bring the error down to a constant?  
 say  $\frac{1}{4}$

But SAT is in EXP or EXPTIME.

1. n. ch. P[ success ] in a single trial



So, can we improve the probability of success. So, one standard trick that is employed in the case of randomized algorithms is to repeat the algorithm many times and this is also sometimes called this technique is called boosting. We repeat the algorithm many times. So, let us see what happens. So, in this case what should you do like let us say you repeat it 2 times you meaning you first randomly generate an assignment and then you again randomly generate an assignment and evaluate fired both these assignments.

So, if both of these lead to the conclusion both of these are;  $\phi$  is evaluated to true then we say  $\phi$  is satisfiable both of them say evaluate to false then we say it is not satisfiable. But what if one of them evaluates true and one of them evaluates to false well if even if one of the assignments lead to  $\phi$  being evaluated to true that means that's a satisfying assignment. So, even if one of them is true then we will declare that  $\phi$  is satisfiable.

So, just think about this. So, even if. So, again what how do we report true or like whether the  $\phi$  is satisfiable or not we declare think about why this is this works. We declare  $\phi$  is in SAT if  $\phi$  evaluates to true in even one of the are trials. So, you can think of this as trials and even if one of these trials lead to  $\phi$  evaluating to true then we say  $\phi$  is satisfiable. So, when do we say that  $\phi$  is not satisfiable if all the  $r$  trials lead to  $\phi$  evaluating to false.

So, this is the algorithm let us try to see what happens in this case what happens let us say we try this algorithm  $r$  times what is the probability of success? The probability of success well it is easier to compute the probability of failure or the probability of error when does the error happen? So, notice what is going on here whenever we declare that the formula is satisfiable we will always be it is always a correct answer.

Because we never declare a formula is satisfiable unless we have found a satisfying assignment. Unless we find the correct assignment we do not declare the formula satisfiable. So, whenever we declare it is satisfiable it is satisfied. The error happens when this formula is satisfiable the error happens when we report the formula to be not satisfiable despite this being satisfiable. So, the probability of the problem is when the formula is incorrectly reported as not satisfiable.

So, rather it is a false negative. So, right now we are in the middle of the pandemic we have the terms false positives and false negatives must be familiar to all but still. So, false negative means that it is incorrectly reported as negative. So, it is actually a positive case or satisfiable formula which is reported as non-satisfiable. So, what is the probability of that the probability of that is not picking the correct solution in any of the  $r$  trials.

So, it should not pick the correct solution the first trial it should not be in the second trial and. So, on and as we have said we declare it as not satisfiable only when it evaluates to false in all the  $r$  trials we declare it as satisfiable if it evaluates 2 in even one of the trials. So, what is the probability of the false negative? It is evaluating as not satisfiable in all the  $r$  trials. So, which is for that to happen in one trial it is  $1 - 2^{-n}$   $1 - 2^{-n}$  by  $2^{-n}$  as we saw here.

I will use another colour as we saw here but here we are repeating this  $r$  times and  $r$  times the all the times the problem the random bits chosen are independent random bits. So the probability of error is every time it gets multiplied. So, there is a  $r$  in the exponent here and you may and this is simplified into  $e^{-r}$   $e^{-r}$  divided by  $2^{-n}$  this is because of this simplification or this usual simplification that we do.

You may have learned in your 11th or 12th when you learned calculus that  $1 - \frac{1}{x}$  as  $x$  tends to infinity tends to  $\frac{1}{e}$ . So, similarly when  $n$  is big and  $r$  is big you can approximate this with  $e^{-r/2^n}$  this is something we can do. So, probability of error is  $e^{-r/2^n}$  this is what we achieve by boosting.

Now let us see how effectively or how far we have boosted. So, let us say. So, we want let us say very low probability let us say for us the probability of error should not be more than let us say 0.1 let us say or 0.2, 0.25 let us say  $\frac{1}{4}$  this is the ideal probability of error. Say  $\frac{1}{4}$  is the probability of error we want to bring down the error probability to  $\frac{1}{4}$ . So, we need to reduce this quantity  $e^{-r/2^n}$  to  $\frac{1}{4}$  which means we can do some reverse engineering.

**(Refer Slide Time: 19:05)**

*Boosting*  
 If we repeat the above algorithm  $n$  times,  $\rightarrow$  in  $n$  trials.  
 $\text{Prob}[\text{error}] = [1 - 2^{-r}]^n \approx e^{-r/2^n}$   
 (Note:  $2^{-r}$  is circled in blue)  
 $\lim_{x \rightarrow \infty} [1 - \frac{1}{x}]^x = \frac{1}{e}$   
 (Note:  $\frac{1}{e}$  is circled in blue)  
 Can we bring the error down to a constant?  
 Say  $\frac{1}{4}$   
 But SAT is not in EXP or EXPTIME.  
 i.e.  $\frac{1}{4}$  chance.  $P[\text{success}]$  in a single trial.

So, we can say  $e^{-r/2^n}$  should be  $\frac{1}{4}$  which implies that  $r/2^n$  should be  $\ln \frac{1}{4}$  which means  $r$  should be  $2^n \times \ln 4$  again its natural logarithm where the logarithm is taken to the base  $e$ . As I have said before whenever we use  $\log$  without qualifying the base will be  $2$  sorry base  $e$ . As I have said before whenever we use  $\log$  without qualifying the base will be  $2$  but here I am explicitly writing  $\ln$  and I am explicitly stating that it is natural logarithm which means the base is  $e$ .



In any case this  $r$  when we want to bring down the error to a constant happens to be  $2^n$  times  $\log 4$ . So, this what is this number this is some this is a reasonably huge quantity. So, when we want to bring down the error to  $1/4^{\text{th}}$  we are we are repeating we are having to repeat it  $2^n$  times  $\log 4$ . But anyway what we could have done instead is to just try out the  $2^n$  assignments one by one that would be just  $2^n$  repeating we did not need to do this randomness stuff.

So this is not really much of a big deal SAT we anyway know that as I described SAT is in  $\Sigma_1^P$  or  $\Sigma_1^P$  prime means it is exponential time complexity class. So,  $k$  equal to 1 to infinity  $\text{DTIME}$  to power  $n$  power  $k$  this is  $\Sigma_1^P$  but anyway SAT is in  $\Sigma_1^P$ . So, as I described. So, what is the benefit of this? Again in this particular problem or in this particular formula it happened to be the case that searching for a positive or searching for a correct solution was like only one into power  $n$ .

So, it was like searching for a needle in a haystack. But there could be other problems or there could be other instances of satisfiability itself where the probability of success may be much better. So, this is not a reason to rule out this, this just this happened to be a simple enough example to illustrate hence I took up this. So, what if instead of; so, here the probability of success in one trial was  $1$  divided by  $2^n$  this was the probability of success.

What if the probability? So, this is inverse exponential. So,  $2^n$  is an exponential quantity what if the probability of success was inverse polynomial let us say  $1/n^k$  and power  $k$ . So,  $n$  is the input length also or here it is the number of variables in the SAT in the formula what if it was this where  $k$  is a constant.

**(Refer Slide Time: 22:19)**

NPTEL

$$P[\text{error overall}] \geq \left(1 - \frac{1}{n^k}\right)^{c \cdot n^k} \approx e^{-c}$$

If we set  $c = \ln 4$ , then we get  $P(\text{error}) \leq \frac{1}{4}$ .

RP: RP consists of those languages  $L$  for which there is a Probabilistic polynomial time TM  $M$  such that

$$x \in L \Rightarrow P_x [M(x) = \text{Acc}] \geq \frac{1}{2}$$

$$x \notin L \Rightarrow P_x [M(x) = \text{Acc}] = 0$$

In this case we could have we could have repeated polynomial many times or if the number of repeats was  $c$  times  $n$  power  $k$  using the same one limit one minus one by  $x$  etcetera. We get that the probability of error becomes  $e$  power minus constant when the number of repeats is  $C$  times  $n$  power  $k$  where  $c$  is some constant. So, if we SAT  $C$  to be if we SAT  $C$  to be I think  $\log 4$  natural log of 4 then we get probability of error is equal to  $1/4$ th which is good.

So, again in this case the number of repeats required is constant times  $n$  power  $k$  which is polynomial time. So, again what is the difference here we said that the number of repeats becomes polynomial time if the success probability of success was inverse polynomial. So, when does the probability of success become inverse polynomial? In this case there was only one true assignment amongst many.

So, if the formula was such that we had many true assignments. So, it not only should it be satisfiable it should be satisfiable with many true instances. In such a case such an algorithm may be helpful otherwise it is not very helpful but anyway randomized algorithms. Again as I said again and again SAT is just a motivating example and we will use other it is more widely used for other problems.

But this I want to tell you. So, that you get some idea of the the idea of boosting and in which cases do the probability of success in which cases the probability of success good enough. So, now this leads to one complexity class called RP.

(Refer Slide Time: 24:26)

NPTEL

is a Probabilistic polynomial time TM  $M$  such that

$x \in L \Rightarrow P_x [M(x) = \text{Acc}] \geq \frac{1}{2}$

$x \notin L \Rightarrow P_x [M(x) = \text{Acc}] = 0$

\*  $P \in RP \subseteq NP$ : Think!

\* The  $\geq \frac{1}{2}$  can be replaced by  $\geq \frac{1}{10}$ , or  $\geq \frac{1}{4^k}$ , or  $\geq 1 - \epsilon$ .

\* Error reduction of Boosting by repeats.

\* One sided error: False Negatives

... ..  $P \subseteq RP \subseteq NP$  ... ..

RP simply stands for randomized polynomial time it consists of those languages for which there is a probabilistic turing machine, probability tuning machine is a which is a tuning machine that can make random choices that is what probability during machine means which runs in polynomial time. So, it is must be probabilistic it turns in polynomial time such that whenever  $x$  is in the language  $x$  is in the language it accepts with the probability of at least half.

And whenever  $x$  is not in the language it accepts with the probability zero which means it virtually never accepts. So, again you may see the motivation we notice that this problem the and the algorithm that we just described works efficiently when the probability of success was higher one by inverse polynomial and not inverse exponential. So that is why we need whenever the probability of acceptance whenever  $x$  is in the language the probability of acceptance should be certainly it should be sufficiently high for the machine.

And it is exactly like this whenever a formula  $\phi$  is not satisfiable this algorithm will never accept there is no false negative or there is no false positive. We declare that a formula is satisfiable only when you find a satisfying assignment. So, there will never be a situation where

the formula is declared satisfiable when it is actually not. So, there is no false positive it only has false negative.

And this definition is also saying the same thing whenever  $x$  is in the language it is accepted with the probability of at least half whenever  $x$  is not in the language it is never accepted. So, this is the definition of the complexity class RP. So, again this we will see some some aspects of this class. The first point is that P is a subset of RP anything that has a deterministic polynomial time algorithm is contained in RP, why?

Because you can view the polynomial deterministic machine itself as a as a randomized machine whenever  $x$  is in the language it always accepts. So, the probability of success is actually one whenever  $x$  is not in the language the probability of success is zero. So, you can that is why any deterministic polynomial time decider can be viewed as an RP decider as well. Second RP is this subclass of NP why is this when is a language L in NP.

So, whenever a string is in the language it must have at least one whenever  $x$  is in the language it must have at least one accepting computation all of it all the other things could be reject but at least one accepting computation should be there. In the case of RP what we are saying is that whenever  $x$  is in the language around at least half of it should be accepted the rest we do not care. And whenever it is not in the language everything should be rejecting which is the same for RP as well as NP.

We want all computation powers to be reject whenever  $x$  is not in the language. So, when  $x$  is not in the language it is the same the condition is the same when  $x$  is in the language RP requires at least half to be accepting paths NP requires only one accepting computation. So, that is ok which is this condition is more than met by the RP yes condition. Hence RP is contained in NP. So, this is something that you can think about if it is not clear and you can ask questions over the forums if this is not clear. So, you think about this please.

**(Refer Slide Time: 28:57)**

$x \in L \Rightarrow P_x [M(x) = Acc] \geq \frac{1}{2}$   
 $x \notin L \Rightarrow P_x [M(x) = Acc] = 0$

\*  $P \subseteq RP \subseteq NP$  : Think!  
 \* The  $\geq \frac{1}{2}$  can be replaced by  $\geq \frac{1}{10}$ , or  $\geq \frac{1}{n^k}$   
 or  $\geq 1 - \epsilon$ .  
 \* Error reduction of 'boosting' by repeats.  
 \* One sided error : False Negatives

Co-RP :  $L \in \text{co-RP} \iff \bar{L} \in \text{RP}$ .  $L \in \text{co-RP}$  if  
 there exists a prob. poly time machine  $M$  s.t.

Second point is that this quantity half here this is really an arbitrary quantity to some extent we could replace it with one tenth for instance any constant one tenth let us say 2-3rds, 1-4th anything and it will still be fine we can even replace it with 1 by n power k inverse polynomial like we did above over here. Even then it is ok because the thing is that if you have a machine that accepts with probability 1 over n power k then you can use boosting to achieve 1 over 10 or 1 over 4 as we just saw.

And even the boosted algorithm will use polynomially many trials. So, the overall running time is still polynomial. So, you can go from inverse polynomial to constant to even something like very close to one you can boost up more and more and get very close to one like 1, -1, minus epsilon where epsilon is very very small. Again so, this half is just an arbitrary number but if you look at textbooks maybe you may see 1-4th or one maybe 2-3rds or something for the in the place of half for the definition of RP.

So, do not be surprised if you see some resource that states a different that is a different requirement in this in place of half it is just because you could move from one to the other just by repeating some number of times. The boosting I have already mentioned and it is a one-sided error. So, whenever x is in the language maybe this is best explained by writing a table. So there are 2 possibilities x is in the language and x is not in the language.

And the algorithm says yes and algorithm says no. So, whenever  $x$  is in the language algorithm could say yes possible it could say no because the probability of acceptance is only around half whenever  $x$  is not in the language it could say it cannot say yes it always has to say no. So, this is never possible. So, this is this never happens. So, there is an  $x$  and there is another  $x$  cross mark that looks like an  $x$  but hope you understand.

So, this area is never this outcome is never possible in the case of RP algorithms. So, there are 2 ways to see this one sided error. One is that whenever  $x$  is in the language it could say that it accepts with probability at least half whenever  $x$  is not in the language it; so, whenever  $x$  is not in the language it gives the correct answer. If  $x$  is not if formula  $\phi$  was not satisfiable it will never say it is satisfiable. So, because it is only false negatives whenever  $x$  is not in the language it always says the correct answer.

Another way to see this is so, that is when you look at this grid row wise another way to see it is when you look at this grid column wise whenever the algorithm says yes we know for sure that it is a satisfying the formula is satisfiable. But when the algorithm says no it could be the it could be either case it could be correctly reporting a no instance or it could be incorrectly reporting a yes instance.

So, there are 2 ways to look at it one is whenever  $x$  is not in the language it says correctly and the 2 is whenever the algorithm reports yes it is correct. So, there are 2 ways to look at it and it can be confusing that is why I repeat it 2 3 times already this particular 2 different ways of explaining the same thing can be confusing. So, but then if you think about it a few times it is not really that hard this is something that to think about in the case of RP.

And I think I may be doing going over time. So, I just conclude this lecture now and will continue in the lecture 28. And I will just very briefly summarize what we did we motivated the need for randomness as a way to guess the solutions that are magically guessed by the NP machines and then we saw that it is better if there is some guarantee it will perform better when there is some guarantee of success.

So, when there is a formula that has only one satisfying assignment it was not performing that well but when the formula had had sufficiently many satisfying assignments we saw why it performs well. So accordingly we define RP to the class of languages that for which we have randomized machines which run in polynomial time and which have a reasonable measure of success. So, which where we allow false negatives but do not allow false positive.

So, it is a one-sided error and we also described the boosting that can be used to increase the or reduce the probability of error or increase the probability of success as and this can be done to some extent based on the resources that we have and yeah that was the summary of lecture 27 and I will continue in lecture 28.