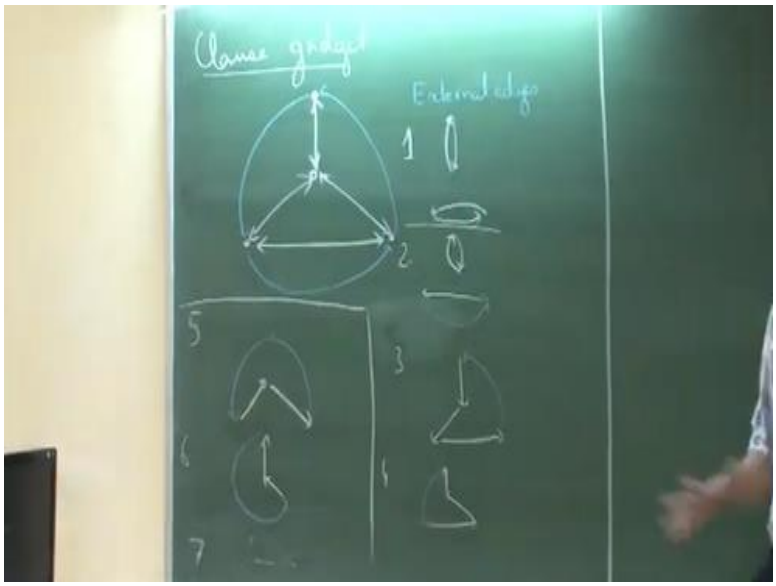
Computational Complexity Theory
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -31
Introduction

So, think I will just start, so first let us just fix this clause gadget thing.

(Refer Slide Time: 00:28)

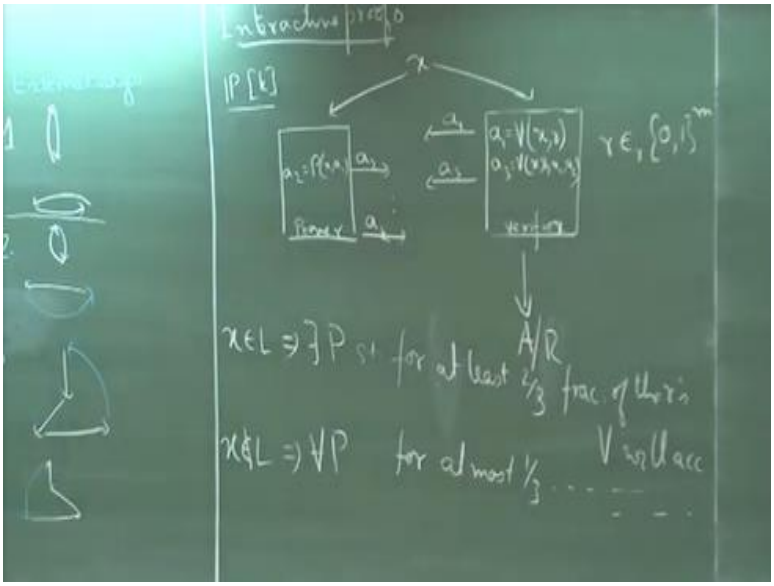


What did I have yesterday is that, and the external edges are what we had earlier. These are the alternative external edges. So, once again, let us see that why this should work. So, suppose if none of the external edges are taken then we have the cycle cover basically by taking this pair and this pair of edges. So, this is for none of the things suppose if you only include the bottom edge so then the cycle cover is so we have this, this and then these two.

So, this is the first cover which has no external edges and this is the second cycle cover which has exactly this external edge so, this is the second one. The third one is suppose if I have to include this one so then I have this edge and this for the fourth case we have this basically, by symmetry, fifth one is suppose I have these two then I take this as my cover. And the last one is you have we have this and this so, now we go this way.

And the seventh one is just symmetric to the sixth, just leave it. And these are the only seven cycle covers in this graph corresponding to these set of things. So, let us come back to interactive proofs.

(Refer Slide Time: 04:04)



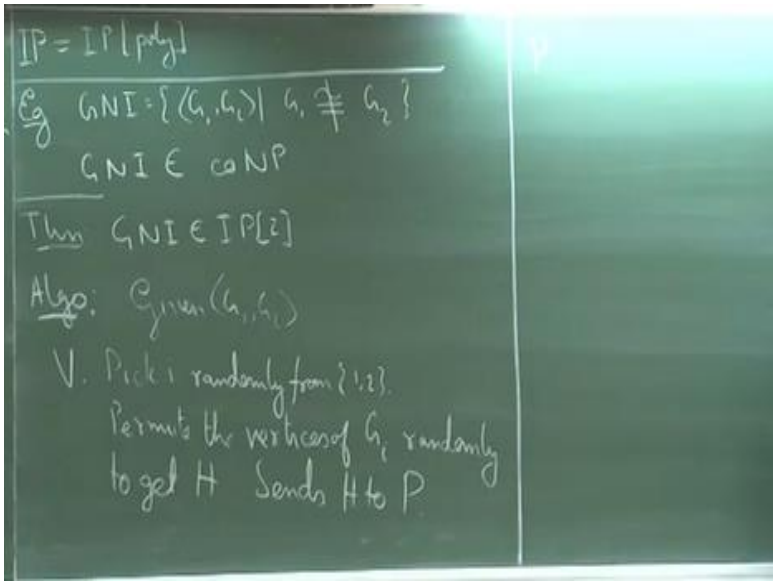
So, we saw this class IP k yesterday. So, basically the model is so there is some prover and there is a verifier. So, given an input x both has access to this input. So, what the verifier does is the verifier computes the string a 1 which is V of x and also, we allow the verifier to be probabilistic. So, it can toss coins so we just denote that by some random string r. So, r is a string in chosen randomly from length, let us say some m.

So, this guy sends its message to the prover now what the prover does is it computes a 2 which is P of x, a 1 and sends it to the verifier. The verifier computes a 3 depending on what x, r, a 1 and a 2 is sends it to the prover and so on finally the prover let us say send some answer a k and based on that the verifier decides whether it would accept or reject the input x. This is essentially what the model is and what do we have that if x does belong to the language.

So, we want to decide if a particular language I mean if x belongs to a language or not so, then the guarantee that we have is that for so there exist a prover such that for at least two third fraction of the r's we will accept. And if x does not belong to L then no matter what the prover function is for at most one third fraction of the r's the verifier will be able to accept. So, this is

what the model was.

(Refer Slide Time: 07:56)



And we said that we will denote this class IP to be IP with polynomial in many round. Of course, I mean it does have a hierarchy the question is whether it is a proper hierarchy. I mean of course IP k defines a certain class of language and IP k + 1 defines another class, but what is the relation? So, we will talk about that. So, actually if k is constant if k does not depend on the size of the input then what we can show is that everything collapses to IP 2.

So, it just basically the verifier asking one question and the prover answering back and based on that the verifier can decide whether to accept or not. I mean it depends no see basically the way you can think of this is maybe to compute the first question the verifier uses let us say the first few random bit let us say the first five random bits of r and then to compute a 3 uses the next ten whatever.

So, ultimately the total size is some polynomial so we can always assume that it is the same r that the verifier has access to. So, whenever we talk about an IP k, we are fixing k so we want the number of rounds to be bounded by sum k. So, the crucial point here is that the prover does not have access to r. In other words, whatever the coin tosses that is being conducted by the verifier that remains hidden from the prover.

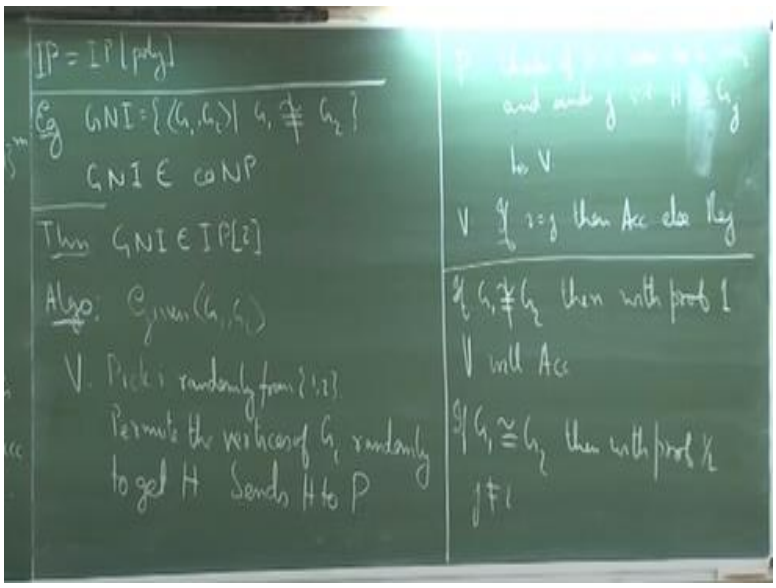
So, we will soon see why it is so important. So, let us look at an example. So, consider the following language, the language of graph non isomorphism so, you are given two graphs G 1 and G 2 as your input and you accept this if G 1 is, this is the sign for not isomorphic to G 2. So, 38 guys, you saw this question in your 340 courses that the graph isomorphism problem is in NP. And that is not very difficult to show, because the way you argue it is in NP is you just guess a permutation.

So, if G 2 is that permutation operated on G 1 then of course you accept otherwise not. So, if the graphs are isomorphic there exists some permutation otherwise that does not take. So, that immediately implies that graph non isomorphism is in co NP. So, can we say something more so can we give a interactive proof model for graph non isomorphism. So, as it turns out what we can show is that graph non isomorphism is in IP 2.

So, in 2 rounds of an interactive proof, you can decide graph non isomorphism. And the algorithm is quite simple as well as the analysis. So, the algorithm is that given G 1 and G 2, what the verifier does first is it tosses a coin and depending on whether it is head or tail it chooses an i from the set 1, 2. So, take i randomly from 1, 2 and then permute the vertices of G i again randomly to get let us, say a graph H.

So, what the verify does is? It tosses a coin suppose if it is head it will choose i to be one and if it is tail choose i to be two. And then depending on what i is it will permute the vertices of G i. So, if it comes out head it will permute the vertices of G 1 otherwise it will permit the vertices of G 2 to get a graph H. So, clearly H is isomorphic to G i since it is just permitting the labels of the word of the vertices of G i. And then it sends H to the prover.

(Refer Slide Time: 14:31)



So, what the prover does given H is? So, prover has infinite computational power, so what he does is he checks if H is isomorphic to G 1 or G 2 and sends j such that H is isomorphic. He sends j such that H is isomorphic to G j to the verifier. Now what the verifier does is? If i is equal to j then we will accept else we will reject, so that is the complete protocol. So, now let us see that suppose if we have a graph G 1 and G 2 which are not isomorphic.

So, if G 1 is not isomorphic to G 2 what happens? What can we say? So, if G 1 is isomorphic to G 2, then with probability 1 we will accept. Because the graphs are not isomorphic, so there is no way in which I mean the prover can always find that out. But the problem is if the graphs were isomorphic so if G 1 let us see is isomorphic to G 2, then what happens? So, then in this case H is isomorphic to both G 1 and G 2 and the prover finds that out that H is isomorphic to G 1 and G 2 and then he just sends back some j.

So, the probability that j will be equal to i is exactly half. So, if G 1 is isomorphic to G 2 then with probability half, j is not equal to i. But now what we can do is if you want because we want to boost this probability, we can take repeated trials I mean what the verify does is it finishes one round so, then it again does the same thing. So, then again computes a random I mean again it does a random coin toss to get an i, it generates a graph H and it again asks the prover which of the two graphs is this isomorphic.

And depending on the answer it again outputs. So, if at even in one of the cases the verifier gets that i is not equal to j then he knows that the graphs have to be isomorphic, because the prover has been fooled. So, let us say he takes t trials the probability that in none of the t trials the prover will be fooled is 1 over 2 to the power t. So, the probability that in at least one of the t trials the verifier gets the correct answer is 1 - 1 over 2 to the power t.

So, that is sufficiently high probability. I mean, you can say that if both of them are isomorphic heel always send one you can just fix that, does not matter. Because he does not have any choice I mean, he cannot make out which graph the verifier had permuted to come to generate age, so might as well can just send it. So, we assume that the prover is a deterministic machine so we will always assume that he sends one in that case, does not matter, verifier can send what?

No, I mean the graphs are always I mean the prover has access to both the graphs. So, that is the point, so now the goal so what is the goal of the prover? So, again here we are assuming that let us say that I am the verifier and you are the prover. So, you want to convince me that these two graphs are not isomorphic. So, suppose even if the input is that the two graphs are isomorphic, you can always tell me that know these two graphs are not isomorphic and I have no way to find that out.

Because my computational powers are limited so I am handicapped at far as computational powers are concerned, so I cannot find that out. Which one? Because he wants to hide the fact from the prover that which of the graphs is H isomorphic to. So, let me just post the question in a different way, so what you are asking, let me just pose it in a different way, what if the prover had access to this coin toss? In other words, what if the prover could see what is the result of the coin toss?

In that case, he can easily fool the verifier I mean, if the verifier let us say comes up with one then the prover knows that ok. This guy is going to permute the vertices of G 1 to construct it so I might as well output j is equal to 1. And if we gets the thing as till just the other way around. So, here it is very important that this coin toss is a private coin toss I mean the prover does not have access to whatever is the random bit that the verifier is using because otherwise you can

fool the verifier.

Again, if you knows this random string then he can just compute the inverse of that permutation and get back the original graph and then find out which of the graph he had permitted. Because the prover's goal is always to accept whatever the input is. He does not care, but the verifier is skeptical the verifier will not allow a wrong input to get accepted. So, you can send it at the same time also does not matter.

So, you can let us say take you can make t point losses here, you can generate H 1 to H t send them all to the prover and but now some of the graphs will be some of the H i's will be isomorphic to G 1 and some of them will isomorphic to G 2. The verify knows which is isomorphic to what because he had toss the coins, so he has all that information. So, now depending on what the answers so he can send a vector of values.

But he cannot send his random bits, he can only send this graphs that he is constructed. And then depending on what the prover is doing for each of those graphs, the verifier will finally be convinced with very large certainty. So, again the point is that how these coin tosses are being conducted I mean, whatever is the random string that the verifier uses is it a public random string or a private random string does make a significant difference.

So, is this clear to everybody this protocol and the idea behind it? Any questions? So, IP 1 means that the prover cannot say so it will be BPP I would guess. Because what does IP 1 means? IP 1 means that the verifier is just sending a string but it cannot get any answer back from the prover. So, please just sending a string, it is same as doing nothing. So, it is just a probabilistic machine sitting over here so it is the same as BPP.

So, if it does not permit at all so if you just sends so the goal of the prover is to make the verifier always accept. So, he will just check that which of the two graphs this is equal to. So, if the verifier had sent G 1 the prover will just send 1 back to the verifier and if the verifier had sent G 2 then the prover will just send 2 back. So, it always the verifier always ends up accepting. Oracle will always supposed to return the correct so oracle is totally different, oracle is in terms

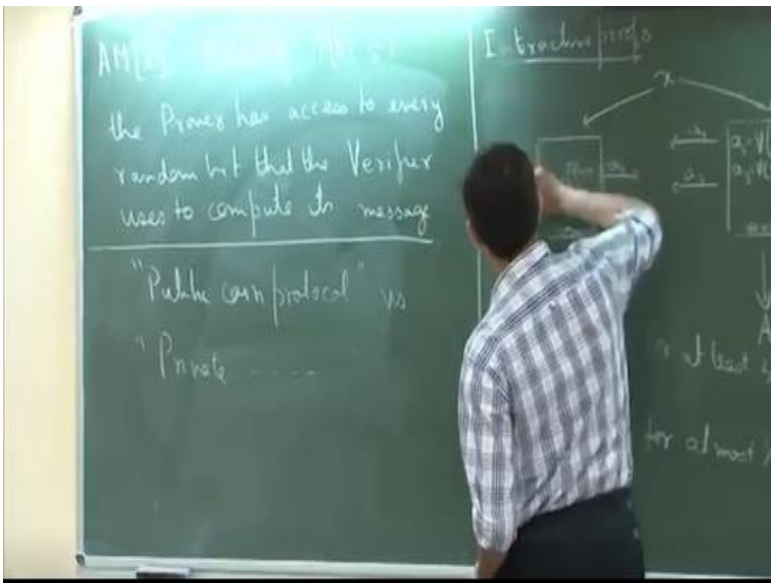
of a language.

So, there is some language which is your oracle and you are just asking whether a string belongs to that language. So, the oracle does not have any hidden motive to make the fine. Oracle is a black box, but it does not have any hidden motive. So, here the motive of the prover is to always make the verifier accept, no matter what the input is. And the goal of the verifier is to be skeptical and only accept those things which are in the language.

But oracle does not operate with any such agenda. So, as somebody said that it always gives the correct answer. SAT bar is co NP so co NP is not known to be in IP 2. I mean in a sense that I think if co NP is shown to be in IP 2, then the polynomial hierarchy will collapse. So, we will see that we will see a more results about these classes in the coming few lectures. But it is very important that the model is clear to everybody as to what we are dealing with here.

So, now that we saw this thing about private coin tosses let us ask the question that what happens if the random variables are accessible to the prover as well.

(Refer Slide Time: 27:25)



So, we define the class AM, so AM stands for Arthur-Merlin. So, AM k is the class of all languages that belong to IP k. So, it is a subset of IP k such that the prover has access to every random bit that the verifier uses to compute its message. So, there is a subtle point here, the point

is that here we had assumed that the entire random string is available to the verifier. But the point is that so see the way I phased this definition the prover has access to every random bit that the verifier uses to compute its message.

So, when it is computing the message a 1 whatever other random bits that the verifier uses to compute a 1 the prover has access to only those random bits and not the rest of r. So, again in the next round again when it is computing a 3, let us say it uses some other set of random bits, again the prover will know what those random bits are. So, in other words, there is no screen between the prover and the verifier in this case.

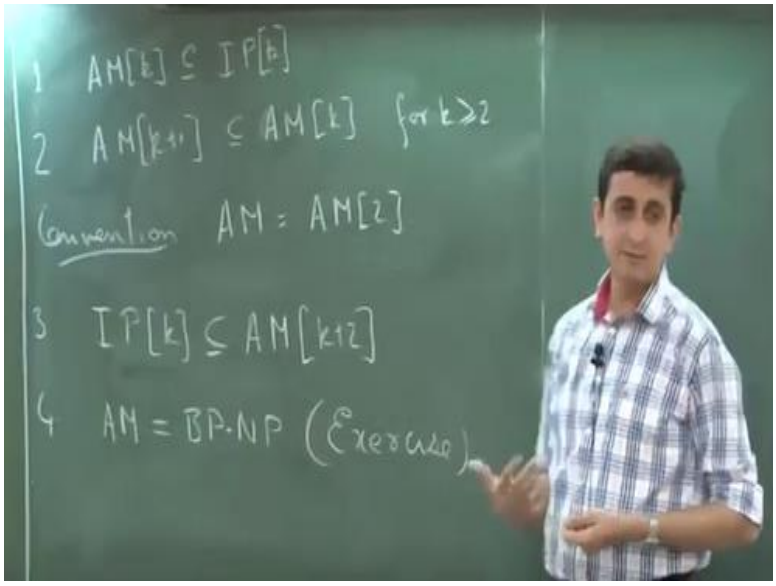
So, whatever are the coin tosses that the prover, I am the verifier makes the prover can see the answer to those coins. He can remember so both of them can remember earlier coin tosses. Because they are, I mean, the prover has unlimited computational power. So, he can remember anything and do anything. Even the verifier can remember earlier coin tosses because it is a polynomial time machine.

So, you can store those bits in some table. But the point is that now this kind of a thing cannot happen I mean, I cannot say that graph known as (()) (30:38) is in AM 2. But as it turns out that that is also true but by our totally different argument and you see that probably on Friday that graph non isomorphism can also be shown to be in AM 2. You mean in the art thermal? You mean in this protocol? In this model, it does not know here is does not know, here basically everything is hidden from the prover. whatever V is doing, V is doing it secretly.

Yes, in a name, it will know I mean if this way to be an AM model then what is the permutation. Exactly, so that is why we need a completely different protocol to be able to show that graph non isomorphism is in AM 2. This will not, so that will see in the next class. But this is the basically there are the differences what is known as public coin protocol versus private coin. So, in one case the prover has access to the random bits of the verifier and in the other case, it does not.

So, let us look at some properties of these classes. Some of them are easy to see but others will prove later on.

(Refer Slide Time: 33:19)



So, of course by definition AM k is contained in IP k. What is also known as this is a more non-trivial is that AM k +1 is contained in AM k for k greater than or equal to 2. So, any Arthur Merlin model with some constant rounds of interactions can be brought down to 2 rounds of interaction without any loss of generality. So, this is more non-trivial. So, you can try this out but let us see we will probably discuss a proof of this maybe after, this always is the number of rounds.

So, how did I define a round? So, a round is a message, does not matter actually. So, therefore this class AM so this is mode of I should not state this as a property, but this is a convention. So, when whenever we say AM, this refers to AM 2. So, the verifier makes some coin tosses sends a query to the prover, the prover does some computation sends back its answer to the verifier and then again it does some probabilistic computation and depending on what that answer is it is either accepts or rejects.

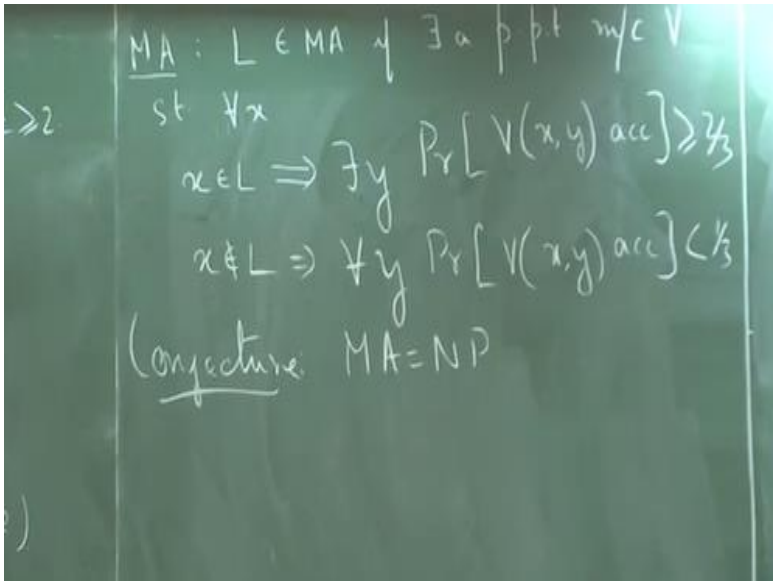
That is what the language AMS. Basically, not for this reason, but it is because of this property. So, the third property about these classes is that AM k is contained in IP k. By definitions IP k is contained in AM k + 2 and this is again something non-trivial and we look at a proof of this later on. So, because of this again any constant labels of the IP, I mean any constant rounds of the IP protocol can be brought down to two rounds.

And this maybe you can try. So, what again is known as AM is equal to this class BP dot NP. So, this I will leave as an exercise. So, recall what the definition of this class was so it is a BP operator prefix to this complexity class NP. So, this just try as an exercise, so let me mention this. So, why was this term Arthur Merlin point for this class? So, again that is because of some historical region.

So, in medieval times in England, there was some king called Arthur and he had a court magician by the name of Merlin. So, the idea here is that the king will ask Merlin some questions and Merlin is all always able to give the correct answer, no matter what the king asks and whatever point losses that the king makes so, they are also not hidden from Merlin. So, when this class was first introduced so this was in 87 by Lacy Babai.

So he had that model in mind and that is why he gave this name. So, what if we just change the protocol a little bit and instead of Arthur first asking the question what if Merlin decides to send some string to Arthur first. And then depending on that Arthur would accept or reject.

(Refer Slide Time: 38:53)



So, that is what this class MA is which is Merlin Arthur. So, Merlin Arthur is how do we define this? So, one way to define this is using the interaction model, but there is also a other way to define it, let me use that. So, L is said to be in this class MA if there exist a probabilistic

polynomial time machine V such that for all x belongs to L implies that there exists some certificate y, basically the certificate that Marlin sends to Arthur such that with high probability Arthur would accept, so that is with probability z V given x, y and z.

So, this is a probabilistic machine so I might not as well include this. So, V given x, y accepts this one third. I mean again, so here we just have a 2 round so just or maybe just 1 round. Because Merlin sends a string to Arthur and then Arthur decides to accept or reject. So, MA k would probably be contained in AM k + 1 or something like that. Exactly so if you look at this definite definition of MA it is the same as that of NP, but the verifier is a probabilistic machine.

And actually, it is a conjecture so this is a very widely believed conjecture in complexity that MA is probably same as this NP. So, again, BPP is contained in MA because given an x if I mean if it just does not use the answer that Merlin sends Arthur so without using y this machine is just a BPP machine. So, by definition MA contains NP as well as BPP. But it is also conjecture that these two classes are the same. Which one?

NP dot BP does not make sense now I mean, what do you mean? BP dot NP is separate because BP dot NP is again seemingly more powerful because what is happening in AM 2 is first the verifier does something then it asks a question and then depending on the answer it decides to accept a reject. But here this is I mean this clearly has lesser power because this I mean this y does not depend on a question that the verifier is just some string which Merlin decides to send to Arthur. Verifier is always probabilistic.

So, we are always dealing with so what is NP dot BPP, how do you define that class? So, BP dot C was a there exist dot BPP. But that is why you are using the fact that it is a BPP machine. So, what is there exist dot C? So, if x belongs to L then there exists a y such that this is a should be, I mean, I do not see anything wrong in that maybe you can call it as there exist dot BPP. You mean if this is true then, yes.

If that is closed under polynomial probabilistic, polynomial reduction, then does it imply this that I do not think I am done maybe but that is more difficult, I do not say how that gets implied. It

does, because I mean, he has access to x so depending on what x is he sending a string y so he does have. But the question is that his y is not an answer to some question which the verifier asks. It is basically like a certificate. So, we will stop here.