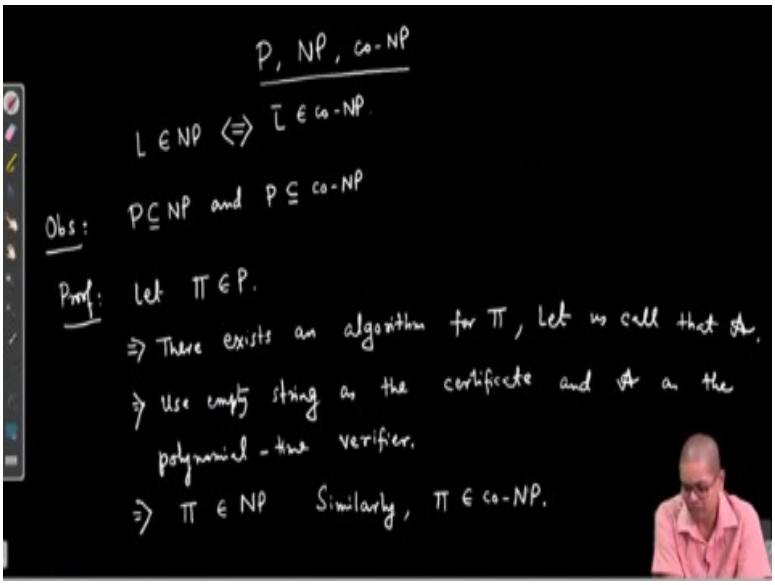
Selected Topics in Algorithm
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 32
Turing Reduction, Karp Reduction

Welcome, so, in the last class we have started NP completeness and this complexity class we will continue that in this class. So, we have seen three main complexity classes P, NP and co-NP.

(Refer Slide Time: 00:55)

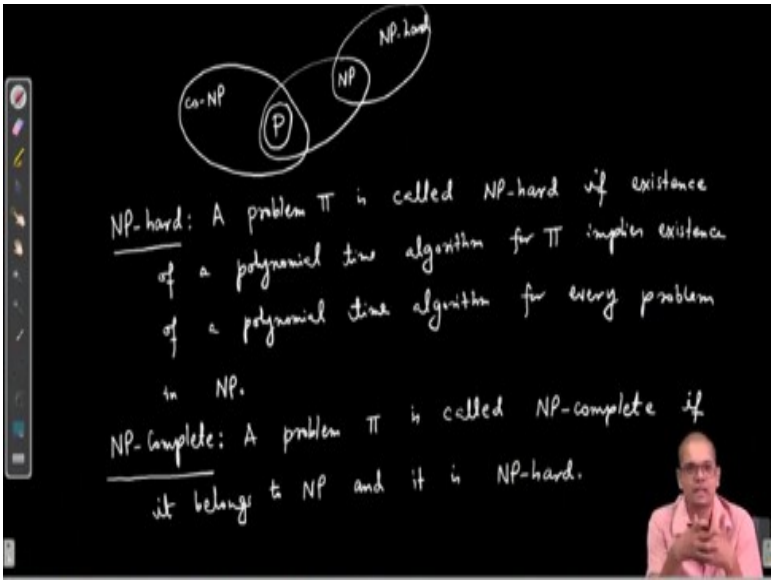


And we have seen that L belongs to NP if and only if L bar belongs to co-NP. Now what is their relationship with P? So, we claim that this is also obvious that P is a subset of NP and P is a subset of co-NP this is an observation. So, why? First prove that P is a subset of NP, for P to be a subset of NP for instances of a problem. So, let Π be a problem in P now I need to show that Π belongs to NP.

That means for YES instances there should exist a polynomial verifier and a polynomial size certificate which can verify that the instance indeed a YES instance. So, because Π is in P this implies there exists and algorithm for Π. Let us call that A. Now we claim that use empty string has the certificate and A as the poly time verifier, polynomial time verifier, A can simply solve this problem an instance of Π and check whether the instance YES instance or NO instance.

It does not need any certificate so this implies that Π is in NP. Similarly, Π is in co-NP because the same thing the same algorithm A can verify in polynomial time that the input instance is indeed a NO instance hence this algorithm A perfectly fulfils the requirement of a polynomial time verifier for verifying the known YES of the instances. So, this concludes the proof.

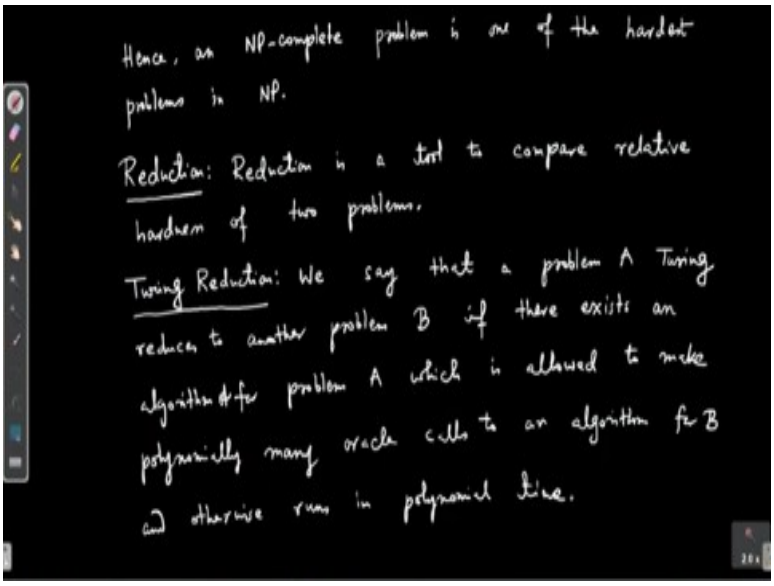
(Refer Slide Time: 04:44)



So, what now how does the diagram look like. Here we have this complexity class P and here the complexity class NP and here is the complexity class co-NP, P is a subset of NP intersection co-NP. Now let us introduce a notion called NP hard, a problem Π is called NP hard if this requirement is fulfilled. If existence of polynomial time algorithm for pi implies existence of polynomial time algorithm for every problem in NP that means you know NP hardness is like a harder.

This is NP hard means if there exists a polynomial team algorithm for any problem which is NP hard then that means that all problems in NP can be solved in polynomial time. And what is NP completeness? So, NP completeness is a problem pi is called NP complete if it belongs to NP and it is NP hard. So, in some sense NP completeness in an NP complete problem is the hardest problem within the class NP.

(Refer Slide Time: 08:56)

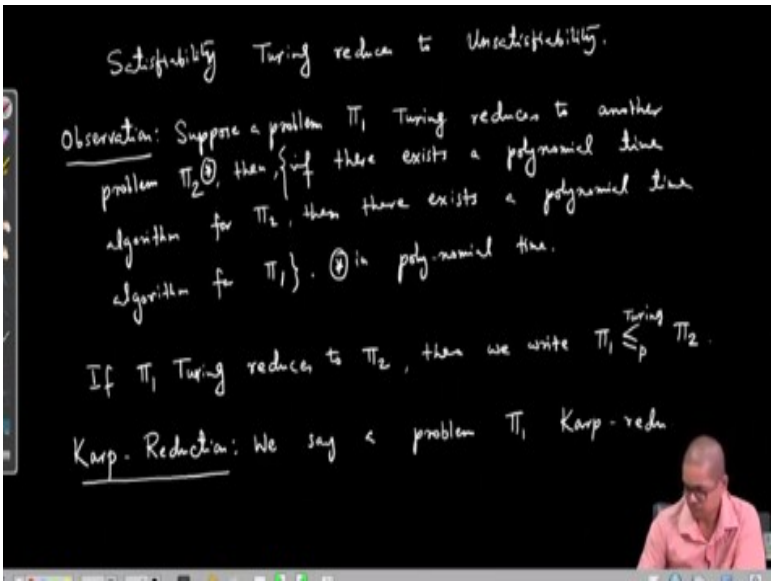


Hence, an NP complete problem is one of the hardest problems in NP. Now how can we show some problem is NP? How can we compare relative hardness of two problems and for that we introduce the notion of reduction? Reduction is a tool to compare relative hardness of two problems. There are two kinds of reduction one is called Turing reduction. We say that a problem A Turing reduces to another problem B, if there exists an algorithm for problem A which is allowed to the resistant algorithm.

Let us call algorithm A for problem A which is allowed to make polynomially many oracle calls to and algorithm for B and otherwise runs in polynomial time. So, let us see. So, a problem A we say it Turing reduces to another problem B. If suppose there exist an algorithm for B and that algorithm we; are allowed to make oracle call to that algorithm. With that algorithm will solve instances of B and this is a Turing direction from problem A to problem B.

If there exists another algorithm say Cal A for problem A which solves means which makes at most polynomial mini oracle calls to algorithm for B. It calls uses the algorithm for B as a subroutine and other operations other normal operations it makes polynomial other normal operations.

(Refer Slide Time: 14:43)



So, for example satisfiability Turing reduces to unsatisfiability. It means given a Boolean formula the question for satisfiability is that is it a YES instance or NO instance. And suppose someone gives me an algorithm for unsatisfiability now make just making one call to this algorithm as a subroutine I can check whether my any whether the input for the satisfiability instance is a YES instance or NO instance.

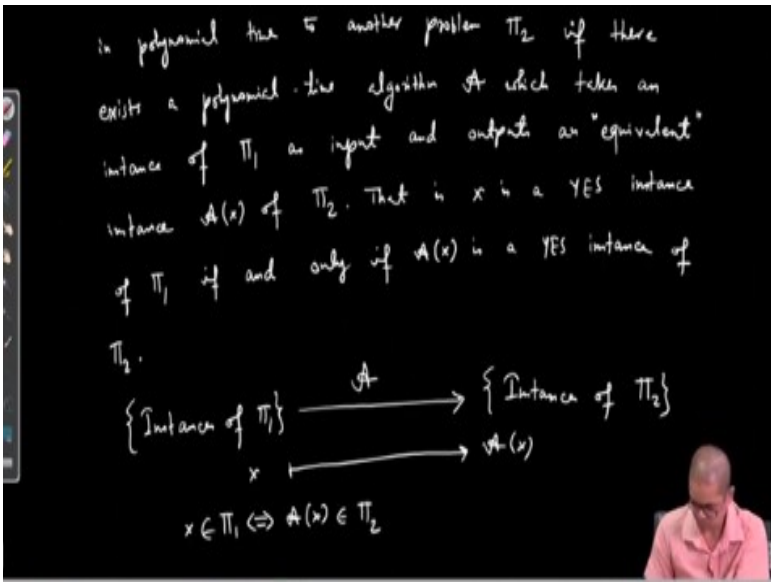
An input for the satisfiability instance is a YES instance if and only if the in that same in same formula given as input to the unsatisfiability instance is a NO instance. So, there is an observation. Suppose of problem Π_1 Turing reduces to another problem Π_2 then if there exists polynomial time algorithm for Π_2 then there exists polynomial time algorithm for Π_1. So, with this sentence within braces as a whole.

So, suppose Π_1 Turing reduces to Π_2 then existence of a polynomial time algorithm for Π_2 implies existence of polynomial algorithm for Π_1 that is directly from the definition. So, if Π_1 Turing reduces to Π_2 then we write Turing reduces to another problem Π_2 in polynomial time the reduction itself works in polynomial time. You know in this course whenever we are doing NP completeness, we usually will be dealing with polynomial time reduction.

So, if not mentioned otherwise it is typically assume that no reduction should run in polynomial time then we write Π_1 reduces in polynomial time here you write Turing Π_2. So, this is the first

kind of reduction. The second kind of reduction is what is called Karp reduction which is more popular.

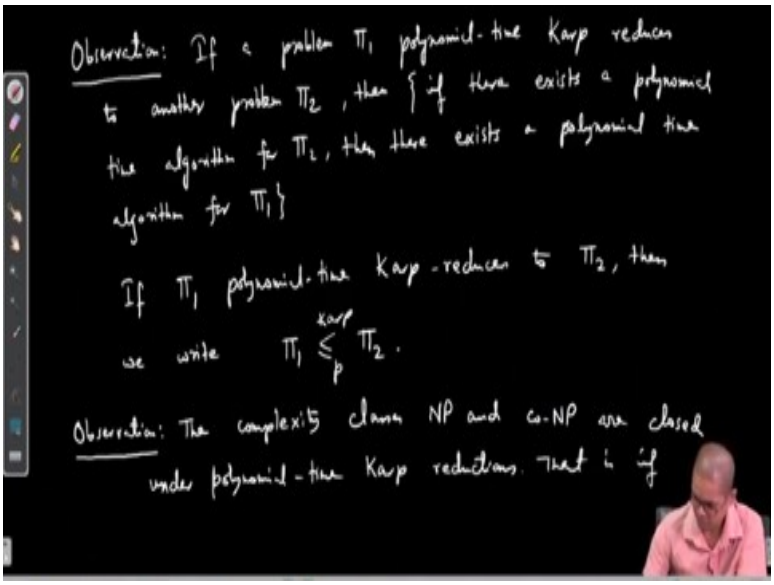
(Refer Slide Time: 20:25)



We say a problem Π_1 Karp reduces in polynomial time to another problem Π_2 if there exists polynomial time algorithm A which takes an instance of Π_1 has input and outputs and equivalent instance A of x of Π_2 that is equivalent means that is x is a YES instance of Π_1 if and only if x is an YES instance of Π_2. That is here I have instances of Π_1 is set maps to this algorithm takes instances of Π_1 as input.

And outputs instances of Π_2, x is mapped to A of x, x belongs to Π_1 because Π_1 is modelled as a language if and only if A of x belongs to Π_2.

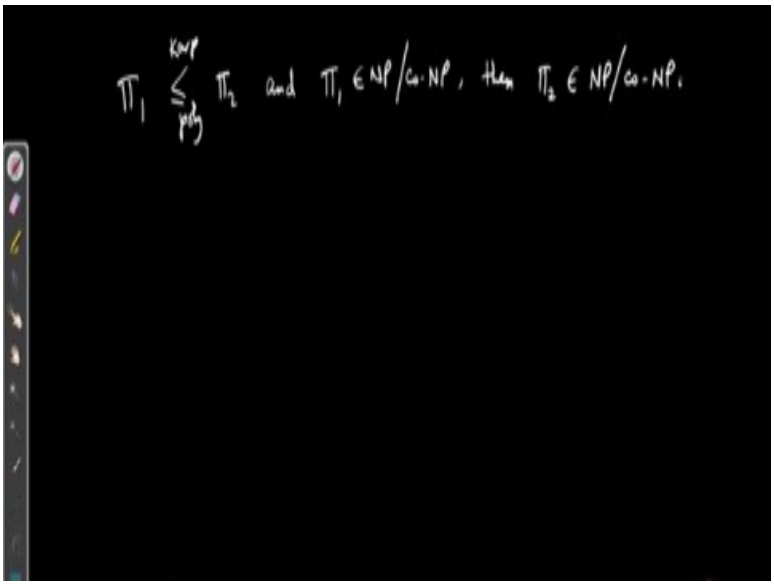
(Refer Slide Time: 24:07)



Again, it is an easy observation if a problem Π_1 polynomial time Karp reduces to another problem pi 2 then if there exists a polynomial time algorithm for pi 2 then there exists a polynomial time algorithm for Π_1 and indeed, so if what is the algorithm? The algorithm for Π_1 simply takes an instance x and runs the reduction Karp reduction Cal A and gets the instance A of x.

Now if I have an algorithm for solving Π_2 then I can run that algorithm on the instance A of x and the output is x belongs to Π_1 if and only if $A(x)$ belongs to Π_2 because of this. So, in this case we write if Π_1 polynomial time Karp reduces to Π_2 then we write Π_1 reduces to Π_2. Here is another nice observation the complexity classes NP and co-NP are closed under Karp reductions under polynomial time Karp reductions.

(Refer Slide Time: 28:56)



That is if Π_1 polynomial time Karp reduces to Π_2 and Π_1 belongs to say NP or co-NP then Π_2 also belongs to NP or co-NP. So, let us stop here and in the next class we will be seeing concrete problems and concrete reductions. Thank you.