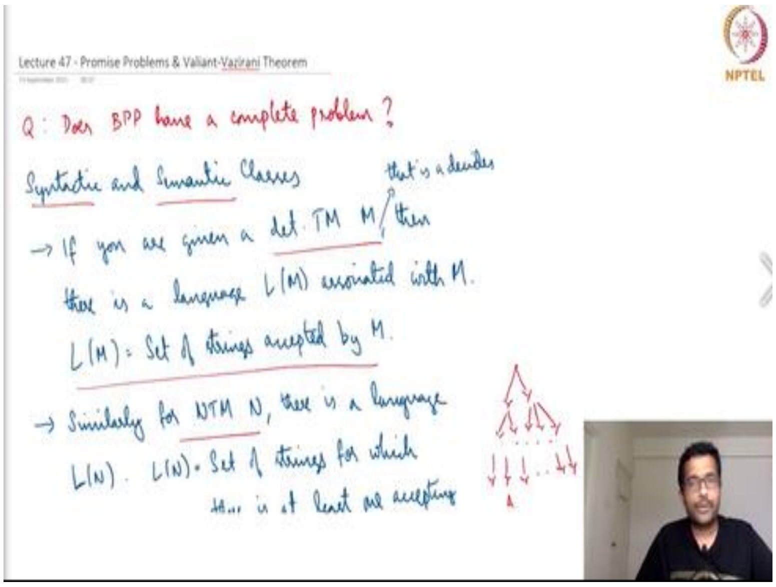
Computational Complexity
Prof. Subrahamanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture - 47
Promise Problems and Valiant- Vazirani Theorem

(Refer Slide Time: 00:15)



Lecture 47 - Promise Problems & Valiant-Vazirani Theorem

Q : Does BPP have a complete problem?

Syntactic and Semantic Classes

→ If you are given a det. TM M that is a decider then there is a language L(M) associated with M.

L(M) = Set of strings accepted by M.

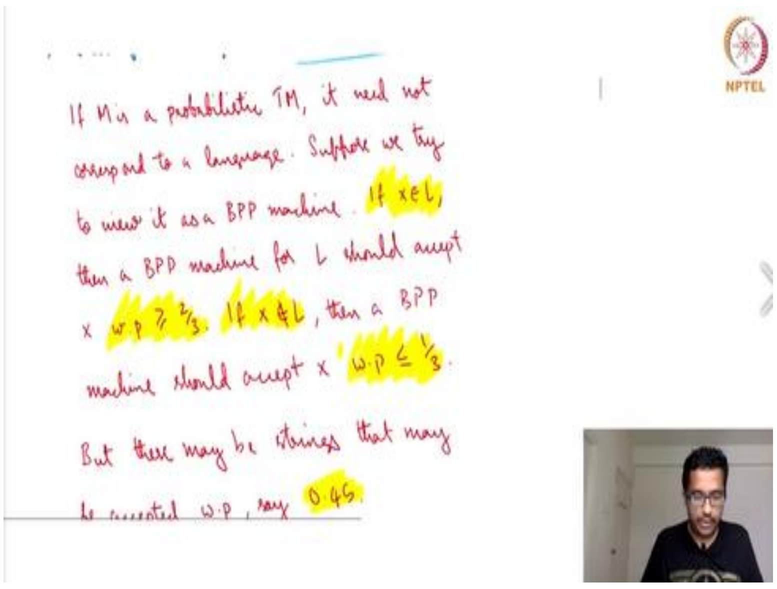
→ Similarly for NTM N, there is a language L(N). L(N) = Set of strings for which there is at least one accepting

Hello and welcome to lecture 47 of the course computational complexity. In this lecture and the next we will see what are called promise problems. Promise problems are a generalization of languages and for a specific promise problem called unique set. We will see the complexity of unique set. So, to begin, let me ask this question. Does BPP have a complete problem? The complexity class BPP.

And this leads to the definition of or two classes called Syntactic and Semantic classes or two class of classes. So, suppose you have a Turing machine M that is a deterministic polynomial time Turing machine. Suppose it is a deterministic polynomial time Turing machine. So, I am only talking about deciders then there is naturally a class of languages or class of strings that it accepts and everything else it rejects

So, there is a language L N all the strings in L M it accepts and all the strings not in L M it rejects. So, given the machine M, its deterministic Turing machine, it is a decider there is a set of strings that it accepts and that that set of strings is the language decided by the Turing machine.

(Refer Slide Time: 01:38)



If M is a probabilistic TM, it need not correspond to a language. Suppose we try to view it as a BPP machine. If $x \in L$, then a BPP machine for L should accept x w.p $\geq 2/3$. If $x \notin L$, then a BPP machine should accept x w.p $\leq 1/3$.

But there may be strings that may be accepted w.p, say 0.45.

Similarly, for a non-deterministic Turing machine we could draw a computation tree like I have drawn on the right-hand side. And we could look at the set of strings that have at least one accepting path, one accepting computation. These are the set of strings accepted by this Turing machine N. This is how a non-deterministic Turing machine decides or accepts a string.

Similarly, we could form a set of strings that are accepted by this non-deterministic Turing machine. We can call it L N which is a class of strings that have at least one accepting path. Now instead of a deterministic or non-deterministic machine what if we are given a probabilistic machine. Let us say we want this probabilistic Turing machine to be a BPP decider.

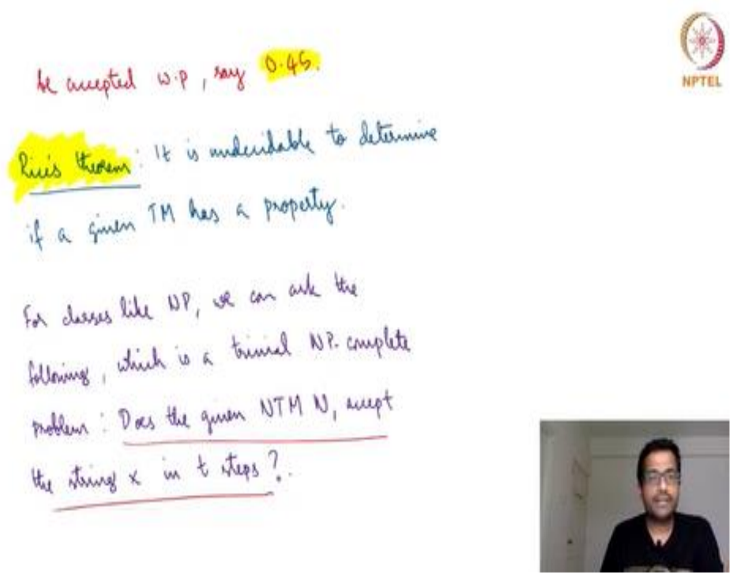
So, given a deterministic or non-deterministic Turing machine whatever be that machine there is a set of as long as it is a decider there is a set of things that it accepts set of strings that does not accept. So, everything that it accepts the rest of it does not accept it there is nothing in between. Whereas if we have a probabilistic Turing machine let us say we wanted to be a BPP decider.

So, you may recall the definition for BPP. Suppose L is in BPP, if x is in the language L. Then the BPP machine for L should accept x with probability two thirds. If x is not in L the BPP machine should accept x with probability at most one thirds. So, there is a gap between the probability of strings, the acceptance probability of strings in the language N and the acceptance and the strings that are not in the language L.

The problem is that if I just give you Turing machine a probabilistic Turing machine it may not meet this. So, what we ideally want is all the strings either should get accepted with probability more than two thirds or get accepted with property less than one-thirds. So, there is nothing in between. It is a BPP decider only then can we say this Turing machine, probabilistic Turing machine is a BPP decider for some language.

But if I just give you a BPP, probable Turing machine it could accept strings with probability say 0.45. Now where do we put that particular string is it supposed to be in the language or is it supposed to be outside the language. So, there is this gap may not be there, because it is just a probabilistic Turing machine. This issue is not there if the underlying Turing machine was a deterministic or non-deterministic Turing machine. So, for a Turing machine to be a BPP decider we need some extra requirements.

(Refer Slide Time: 04:37)



be accepted w.p, say 0.45.

Rice's theorem: It is undecidable to determine if a given TM has a property.

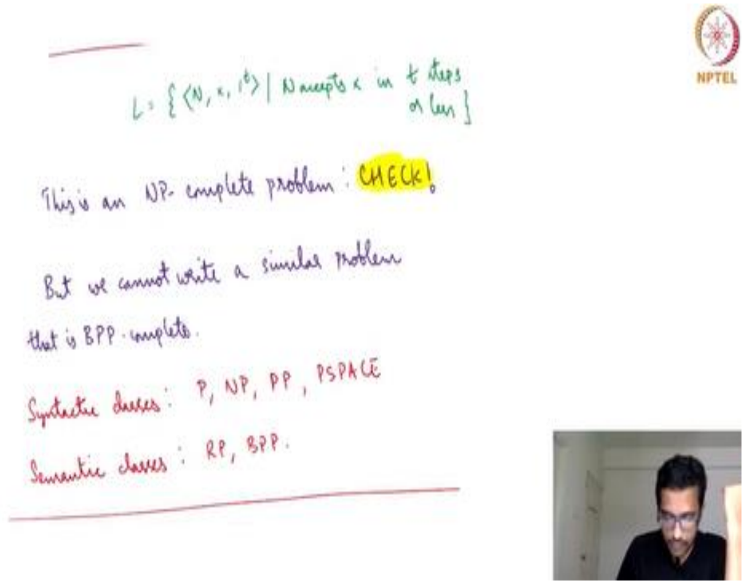
For classes like NP, we can ask the following, which is a trivial NP-complete problem: Does the given NTM N, accept the string x in t steps?.

So, and how do we decide whether a given Turing machine has this property if I give a Turing machine a probabilistic Turing machine and can we say that this corresponds to a language or in other words can we say that this there is a set of strings for which it accepts with property two thirds at least two thirds and all the remaining strings it accepted probability at most one thirds. So, is there such a set of strings?

So, the answer for this is that this particular problem is undecidable. You may have you may recall from a theory of computation you may have seen this theorem called Rice’s theorem which states that it is undecidable given a Turing machine it is undecidable to determine whether it has a certain property. So, in this case the property that we are asking is there some string that gets accepted with probability more than one third but less than two thirds all.

So, this is a property of the Turing machine and it is undecidable to determine that. Rice's theorem was stated for I think deterministic Turing machines but then we can modify it accordingly to show that even this problem is undecidable. So, because of this, given a Turing machine a probabilistic Turing machine we cannot say whether it accepts a BPP language or it accepts a BPP decider for a language.

(Refer Slide Time: 06:09)



$L = \{\langle N, x, 1^t\rangle \mid N$ accepts x in t steps or less$\}$

This is an NP-complete problem: CHECK!

But we cannot write a similar problem that is BPP-complete.

Syntactic classes: P, NP, PP, PSPACE

Semantic classes: RP, BPP.

So, again we started with the question does BPP have a complete problem? So, we are coming to that, so for classes like NP there is a natural very trivial NP complete problem. So, we could simply ask given a Turing machine given a non-deterministic Turing machine and does this not Turing machine N and string x and time t does this non-deterministic Turing machine N accept the string x in time t.

And this is something that it is a class of all triplets N t and x such that n accepts x in t steps. And this kind of trivial NP complete language and you can easily show that this is an NP complete problem. So, I think more formally I think one has to write it like this maybe I will write it here. So, let us say L is a set of all N x and this has to be written in unity so that N accepts x in t steps or less and it can be shown that this is NP complete.

It is very easy to see that it is an NP and basically any other language NP language you can reduce to this. So, if you want to decide if x is in some in language a and propose A is an NP language then you take the Turing machine nondeterministic turing machine that corresponds to a and let us say that is N a and you can input N a and x and a polynomial time and asks whether it accepts.

So, again anyway I leave it for you to check that this is indeed NP complete. The clue is that it is very straightforward or almost told you how to prove it just that the details have to be written down and verified. But the problem is this is an NP complete language, now we cannot write a similar BPP complete problem. Because the problem of deciding whether a given Turing machine is a BPP decider even that is not decidable or not computable.

So, we cannot write a similar problem that is BPP complete. So, as of now it is not clear how to get a language that is BPP complete you can so we will if we go to promise problem there is a way to write a BPP complete promise problem but anyway, as a language a language is just a subset of sigma star where sigma is the alphabet. We cannot write a language that is BPP complete and this brings us to the definition of again I am not writing a formal definition.

But the classification of syntactic and semantic classes. Synthetic classes are those classes for which when you give a Turing machine or Turing machine representation there is an automatically the Turing machine corresponds to a language. Like I said for P, NP etcetera given a Turing machine that runs in deterministic polynomial time there is a language associated with it.

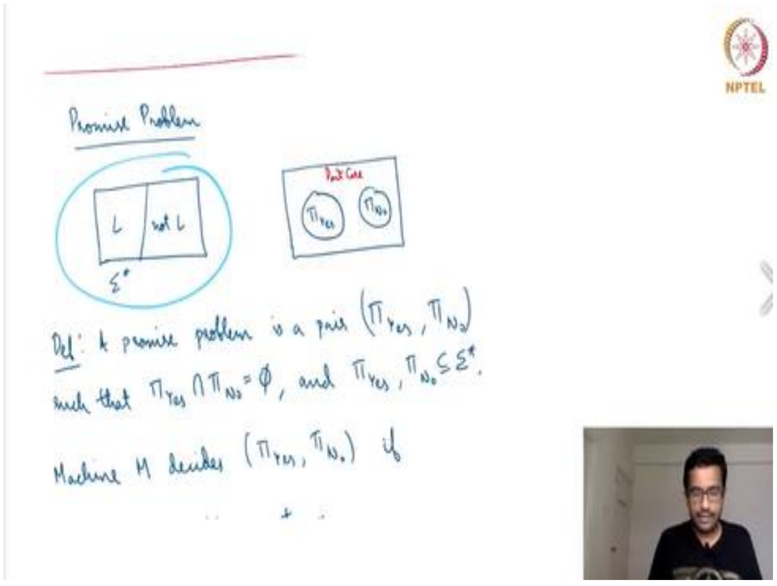
Whereas for RP, BPP given a Turing machine it is not clear whether it is an RP decider, it is not clear whether it is in BPP decider. So, semantic classes are those for which we this a Turing machine by default need not be a decider for that class whereas syntactic any Turing machine that is deterministic will be a decider will decide a language. Same for NP and same for P space and if you think about it, it is the same for PP as well.

Because PP there is no gap so with probably greater than or equal to half you accept and may be strictly less than half you reject. So, any string no matter with whatever probability it is accepted by the Turing machine whatever property it is accepted is either corresponding to it is in the language or not in the language because there is no gap. So, strictly less than half is not in the language and greater than or equal to half is in the language.

So, even PP is a syntactic class. So, syntactic class are those for which a given Turing machine you can immediately look at the Turing machine and tell this is the correct type and so there is a language associated with it. Whereas semantic you cannot do that because in addition to being of the correct type there are some additional restrictions or additional conditions that it has to satisfy. So, as defined, BPP does not have complete languages.

Well, there is unless BPP is equal to let us say turns out that BPP is equal to P then there may be some complete languages.

(Refer Slide Time: 11:28)



Promise Problem

Def: A promise problem is a pair (Π_{Yes}, Π_{No}) such that $\Pi_{Yes} \cap \Pi_{No} = \phi$, and $\Pi_{Yes}, \Pi_{No} \subseteq \Sigma^*$.

Machine M decides (Π_{Yes}, Π_{No}) if

There is something called P complete that we have not seen in this course but there is a notion under a reasonably suitably defined reduction. So, now let me come to the promise problem. What is a promise problem? So, as we saw in the case of BPP we not only required the Turing machine to be probabilistic but we also require that the Turing machine either accepts all strings above a certain probability or below a certain probability and that we needed a gap.

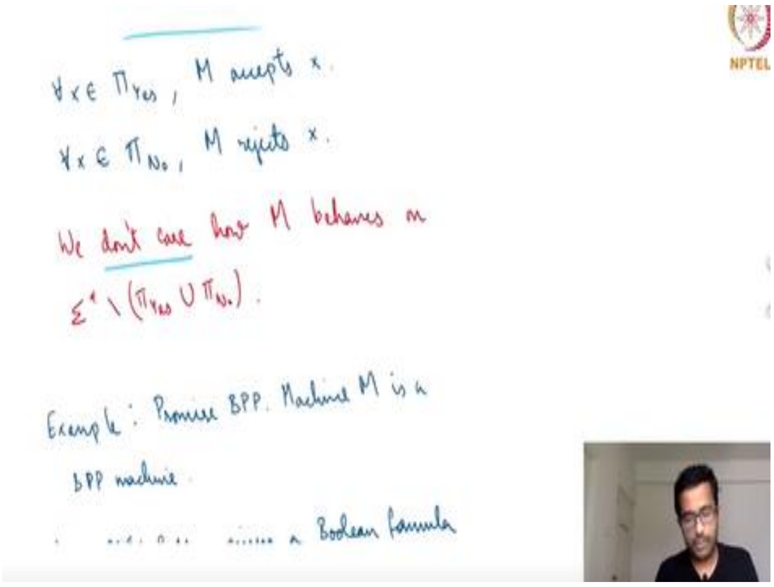
So, this brings us to what is called promise problem. So, we ask the question what do you do when a string is get gets accepted with probability in between? So, promise problem is a way to remove some inputs that are not of interest to us. So, till now we have seen languages and promise problems are generalizations of languages. So, a language is like we have in the left side, it is a subset of sigma star so this entire thing is sigma star.

Everything that is in the language be among with L and everything that is not L is not L. Whereas in the case of a promise problem we have a class of inputs for which we want to say yes and we have another class of inputs for which we want to say no. Obviously these two classes must be disjoint; there is no input for which we want to say both. But the point is that this yes, the set of yes strings and the set of no strings.

These 2 must be disjoint but these 2 need not cover the entire set of strings. There could be strings that are neither in the yes class nor in no class. So, a promise problem is just the classes the yes class and the no class so I have written pi yes and pi no such that two properties are satisfied. First is that they are disjoint pi yes and pi no and second is that both of them are subsets of sigma star.

Which is obviously the case in this figure and it is not a requirement that pi s and pi no together form sigma star there could be other strings that are neither in pi s nor in pi no but we for these inputs we do not care.

(Refer Slide Time: 14:03)



$\forall x \in \Pi_{Yes}$, M accepts x.

$\forall x \in \Pi_{No}$, M rejects x.

We don't care how M behaves on

$\Sigma^* \setminus (\Pi_{Yes} \cup \Pi_{No})$.

Example: Promise BPP. Machine M is a

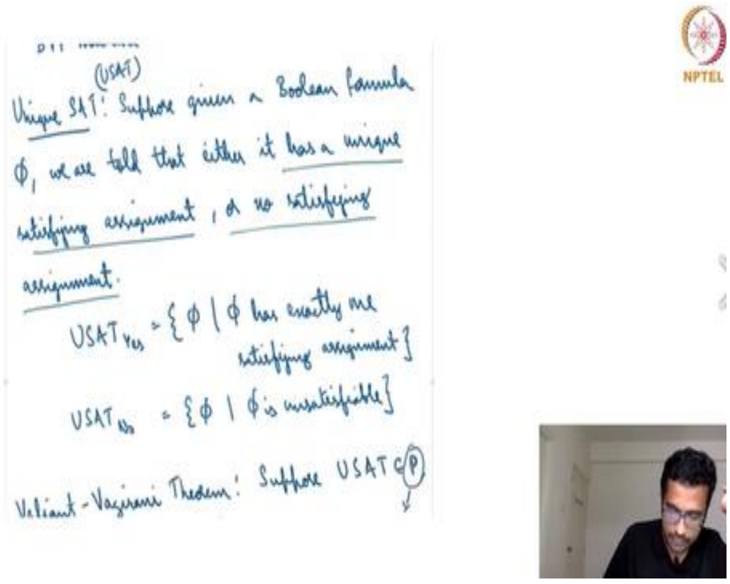
BPP machine.

[illegible] a Boolean formula

So, we say that it Turing machine M decides this promise problem. If it correctly decides on pi yes and pi no. So, for all inputs in pi yes it M should accept, for all inputs in pi no M should reject. We do not care what M does on the other things. So, there is a do not care situation happening here. We do not care how M behaves on the remaining strings there are neither in pi yes nor in pi no. So, we could define something called promise BPP.

Promise BPP are the set of those languages those promise problems for which there is a BPP machine that decides it. And similarly, you can define promise P and define promise NP and so on. So, a promise BPP is a promise problem for which there is a BPP machine that decides it. So, all the strings that are in pi yes, the BPP machine accepts with probability two thirds or more all the strings that are in pi no the BPP machine accepts its probability at most one third.

(Refer Slide Time: 15:12)



Unique SAT (USAT): Suppose given a Boolean formula ϕ, we are told that either it has a unique satisfying assignment, or no satisfying assignment.

$USAT_{yes} = \{\phi \mid \phi$ has exactly one satisfying assignment$\}$

$USAT_{no} = \{\phi \mid \phi$ is unsatisfiable$\}$

Valiant-Vazirani Theorem: Suppose USAT $\in$ $\boxed{P}$

And so just to give one example which we will also see the complexity of consider the problem unique SAT or sometimes it is shortened as USAT. So, this is the class of Boolean formula that have at exactly one satisfying assignment. So, suppose I give you so till now we have seen satisfiability. Satisfiability means we have to given a string given a formula we have to decide whether there is a satisfying assignment or not.

Satisfying assignment, it could be 1 assignment, it could be 10 assignments, could be millions of assignments. We just have to decide is the number of satisfying assignments at least 1 versus is it 0 so is it 0 or non-zero is what we have to determine. In the case of satisfiability in the case of unique SAT the problem comes with a promise. So, this is why again this is why it is called promise problem.

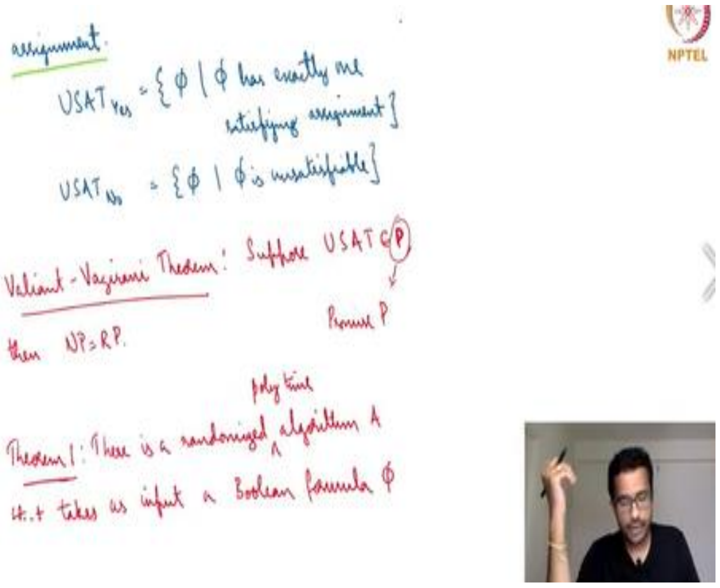
We are told that the given formula either has no satisfying assignment or has exactly one satisfying assignment. So, it do not happen that this problem has this formula has 2 or 10 or 100 or something. It will have either 1 or 0 and then we have to determine whether which of the two cases; does it belong to. So, now you see that it could be potentially simpler because the difficulty inside presumably could have been because there was the satisfiable class itself has so many like.

It could have one, it could have two, it could have thousands, it could have millions and so on that many satisfying assignments. Whereas in the case of unique SAT either it is 0 or 1 we know that the given formula is this or that. Now can this potentially make the problem easier. That is one question that we may ask. So, just to complete the formal definition USAT S is a set of all Boolean formulae that has exactly one satisfying assignment.

And USAT no is the set of all Boolean format that have no satisfying assignment or in other words it is unsatisfiable. So, USAT no is the same as the no part of SAT. USAT yes is not the same as yes part of SAT but it is a subset of that. So, USAT no the same is unsatisfiable. So, we are given this promise that it is this it has a unique satisfying assignment or no satisfying assignment.

So, this is why it is called a promise problem. So, we are when you are given the language, the language comes with the promise.

(Refer Slide Time: 18:03)



assignment.

$USAT_{Yes} = \{\phi \mid \phi$ has exactly one satisfying assignment$\}$

$USAT_{No} = \{\phi \mid \phi$ is unsatisfiable$\}$

Valiant-Vazirani Theorem: Suppose $USAT \in \boxed{P}$ Promise P

then $NP = RP$.

Theorem 1: There is a randomized poly time algorithm A s.t takes as input a Boolean formula ϕ

So, that is why it is called a promise problem. So, you may think perhaps it is easier to decide this problem because it comes with a certain promise and perhaps that it makes it easier. And what we will see in the next lecture is the theorem by Valiant and Vazirani. I think somewhere in the mid 80s maybe 83 I do not know mid 80 sometime maybe I will just say mid 80s. Suppose there was a polynomial time algorithm for unique SAT.

So, when I say polynomial time, we cannot say USAT is in P. Because USAT is not really a language, a language is just a subset. So, the yes instances are in the subset, no instances are not in the subset. So, I should actually say strictly speaking I should say USAT is in promise P not P so it should be promise P. Suppose USAT is in promise P, so promise P is just the promised version of polynomial time.

In that case then NP will be equal to RP. So, NP and RP we already know so this NP being equal to RP is kind of not believed to be true. People think that NP is more powerful than RP. So, what overall what it is saying is that? If USAT has a polynomial time algorithm then NP = RP which is something that is a bit hard to believe. So, we tend to think that USAT is does not have a polynomial time algorithm.

The promise problem of you said is does not have a polynomial time algorithm. So, there is some evidence that even when we restrict to this promise of SAT the Boolean formula that is given as a satisfiability instance. Even when we restricted to the case where the formula either has only 0 or 1 satisfying assignment, not more even then it continues to be hard. So, that is the Valiant Vazirani theorem.

So, in the next lecture we will see the proof of the Valiant Vazirani theorem and just to summarize in this lecture. We saw what is syntactic and semantic classes? So those syntactic classes are those for which a Turing machine automatically corresponds to a certain language. There are semantic it needs to satisfy some other conditions other properties to B e a decider. And BPP is a semantic class because of which there is no immediate BPP complete language.

And then we define promise problem which was a generalization of languages because languages are just a subset of sigma star. Promise problem are like yes instance and no instance but these need not be together this need not be a partition of sigma square there could be some that are neither in the yes or nor in the no instances. We defined unique SAT which is set of all Boolean formula that have exactly one satisfying assignment.

This is the yes unique SAT yes instance and the unsatisfiable Boolean formulas are the no instance. So, and then we said that Valiant-Vazirani theorem says that it is unlikely that even this restricted problem or even this promise problem has a polynomial time algorithm. And with that I conclude this lecture and we will see the proof of Valiant Vazirani theorem in the next. Thank you.